Plan 9™

Programmer’s Manual
Volume 1

Fourth Edition
2002

Computing Science Research Center
Bell Laboratories
Lucent Technologies
Murray Hill, New Jersey
Trademarks referenced in this document:

Plan 9 is a trademark of Lucent Technologies Inc.
Aladdin Ghostscript is a trademark of Aladdin Enterprises.
ARM is a trademark of ARM Limited.
Avanstar is a registered trademark of Star Gate Technologies, Inc.
CGA and VGA are trademarks of International Business Machines Corporation.
Silicon Graphics, IRIS Indigo, IRIS, IRIX, Challenge, and Indigo are registered trademarks of Silicon Graphics, Inc.
Indy and POWER Series are trademarks of Silicon Graphics, Inc.
Ethernet is a trademark of Xerox Corporation.
IBM, PS/2, and PowerPC are registered trademarks of International Business Machines Corporation.
Intel and Pentium are registered trademarks of Intel Corporation.
8088, 80286, 80386, and 80486 are trademarks of Intel Corporation.
Lucida is a registered trademark of Bigelow & Holmes.
Pellucida is a trademark of Bigelow & Holmes.
MIPS, R3000, R4000, and R4400 are registered trademarks of MIPS Technologies, Inc.
R2000 and R6000 are trademarks of MIPS Technologies, Inc.
Microsoft, Microsoft Word and Microsoft Office, and MS-DOS are registered trademarks of Microsoft Corporation.
NFS is a registered trademark of Sun Microsystems, Inc.
PDP and VAX are registered trademarks of Digital Equipment Corp.
PostScript is a registered trademark of Adobe Systems Incorporated.
R2000, R6000, R4000, and R4400 are trademarks of MIPS Technologies, Inc.
SecureNet is a trademark of Digital Pathways, Inc.
Sound Blaster is a registered trademark of Creative Labs, Inc.
SPARC is a registered trademark of SPARC International, Inc.
Unicode is a registered certification mark of Unicode, Inc.
UNIX is a registered trademark in the USA and other countries licensed exclusively through X/Open Company Limited.

Plan 9 continues to grow and adapt. The fourth major release of the system incorporates a number of changes, but the most central is the conversion to a new version of the 9P file system protocol. This new version was motivated by a desire to support files with name elements longer than 27 bytes (the old NAMELEN), but the opportunity was taken to change a number of other things about the protocol, making it more efficient, more flexible, and easier to encapsulate. One simple but indispensable new feature made possible by the protocol change is that the system now records the user who last modified a file; try ls -m to identify the culprit.

Many aspects of system security have been improved. The new security agent factotum(4) maintains user passwords, while secstore(4) keeps them safe and enables single sign-on to multiple domains and machines using a variety of secure protocols and services.

Throughout the system, components have been rewritten and interfaces modified to eliminate restrictions, improve performance, and clarify design. The full list is too long to include here, but significant changes have occurred in a number of system calls (wait(2), stat(2), mount(2), and errstr(2)), the thread library (thread(2)), formatted printing (print(2) and fminstall(2)), security (many pages in section 2, including auth(2), authsrv(2)), and many others.

The changes are sweeping and are accompanied by many new programs, tools, services, and libraries. See the manual pages and the accompanying documents for more information.

Bell Labs
Computing Science Research Center
Murray Hill NJ
April, 2002
A great deal has happened to Plan 9 in the five years since its last release. Although much of the system will seem familiar, hardly any aspect of it is unchanged. The kernel has been heavily reworked; the graphical environment completely rewritten; many commands added, deleted, or replaced; and the libraries greatly expanded. Underneath, though, the same approach to computing remains: a distributed system that uses file-like naming to access and control resources both local and remote.

Some of the changes are sweeping:

- Alef is gone, a casualty of the cost of maintaining multiple languages, compilers, and libraries in a diverse world, but its model for processes, tasks, and communication lives on in a new thread library for C.
- Support for color displays is much more general, building on a new alpha-blending graphical operator called draw that replaces the old bitblt. Plan 9 screens are now, discreetly, colorful.
- A new mechanism called plumbing connects applications together in a variety of ways, most obviously in the support of multimedia.
- The interfaces to the panoply of rotating storage devices have been unified and extended, while providing better support for having Plan 9 coexist with other operating systems on a single disk.
- Perhaps most important, this release of the system is being done under an open source agreement, providing cost-free source-level access to the software.

Plan 9 continues to be the work of many people. Besides those mentioned in the old preface, these people deserve particular note: Russ Cox did much of the work updating the graphics and creating the new disk and bootstrap model as well as providing a number of new commands; David Hogan ported Plan 9 to the Dec Alpha; and Sape Mullender wrote the new thread library.

Other new contributors include Bruce Ellis, Charles Forsyth, Eric Van Hensbergen, and Tad Hunt.

Bell Labs
Computing Science Research Center
Murray Hill NJ
June, 2000

Plan 9 was born in the same lab where Unix began. Old Unix hands will recognize
the cultural heritage in this manual, where venerable Unix commands live on, described
in the classic Unix style. Underneath, though, lies a new kind of system, organized
around communication and naming rather than files and processes.

In Plan 9, distributed computing is a central premise, not an evolutionary add–on. The
system relies on a uniform protocol to refer to and communicate with objects, whether they be data or processes, and whether or not they live on the same machine or
even similar machines. A single paradigm (writing to named places) unifies all kinds of
control and interprocess signaling.

Name spaces can be built arbitrarily. In particular all programs available to a given
user are customarily united in a single logical directory. Temporary files and untrusted
activities can be confined in isolated spaces. When a portable machine connects to the
central, archival file system, the machine’s local name space is joined smoothly to that
of the archival file system. The architecture affords other unusual abilities, including:

Objects in name spaces imported from other machines (even from foreign systems
such as MS–DOS) are transparently accessible.

Windows appear in name spaces on a par with files and processes.

A historical file system allows one to navigate the archival file system in time as
well as in space; backup files are always at hand.

A debugger can handle simultaneously active processes on disparate kinds of hard-
ware.

The character set of Plan 9 is Unicode, which covers most of the world’s major
scripts. The system has its own programming languages: a dialect of C with simple
inheritance, a simplified shell, and a CSP–like concurrent language, Alef. An ANSI–POSIX
emulator (APE) admits unreconstructed Unix code.

Plan 9 is the work of many people. The protocol was begun by Ken Thompson;
naming was integrated by Rob Pike and networking by Dave Presotto. Phil Winterbottom
simplified the management of name spaces and re–engineered the system. They were
joined by Tom Killian, Jim McKie, and Howard Trickey in bringing the system up on vari-
ous machines and making device drivers. Thompson made the C compiler; Pike, win-
dow systems; Tom Duff, the shell and raster graphics; Winterbottom, Alef; Trickey, Duff,
and Andrew Hume, APE. Bob Flandrena ported a myriad of programs to Plan 9. Other
contributors include Alan Berenbaum, Lorinda Cherry, Bill Cheswick, Sean Dorward,
David Gay, Paul Glick, Eric Grosse, John Hobby, Gerard Holzmann, Brian Kernighan, Bart
Locanthi, Doug McIlroy, Judy Paone, Sean Quinlan, Bob Restrick, Dennis Ritchie, Bjarne
Stroustrup, and Cliff Young.

Plan 9 is made available as is, without formal support, but substantial comments or
contributions may be communicated to the authors.

Doug McIlroy
March, 1995
NAME
intro - introduction to Plan 9

DESCRIPTION
Plan 9 is a distributed computing environment assembled from separate machines acting as terminals, CPU servers, and file servers. A user works at a terminal, running a window system on a raster display. Some windows are connected to CPU servers; the intent is that heavy computing should be done in those windows but it is also possible to compute on the terminal. A separate file server provides file storage for terminals and CPU servers alike.

Name Spaces
In Plan 9, almost all objects look like files. The object retrieved by a given name is determined by a mapping called the name space. A quick tour of the standard name space is in namespace(4). Every program running in Plan 9 belongs to a process group (see rfork in fork(2)), and the name space for each process group can be independently customized.

A name space is hierarchically structured. A full file name (also called a full path name) has the form

```
/e1/e2/.../en
```

This represents an object in a tree of files: the tree has a root, represented by the first /; the root has a child file named e1, which in turn has child e2, and so on; the descendent en is the object represented by the path name.

There are a number of Plan 9 services available, each of which provides a tree of files. A name space is built by binding services (or subtrees of services) to names in the name--space--so--far. Typically, a user’s home file server is bound to the root of the name space, and other services are bound to conventionally named subdirectories. For example, there is a service resident in the operating system for accessing hardware devices and that is bound to /dev by convention. Kernel services have names (outside the name space) that are a # sign followed by a single letter; for example, #c is conventionally bound to /dev.

Plan 9 has union directories: directories made of several directories all bound to the same name. The directories making up a union directory are ordered in a list. When the bindings are made (see bind(1)), flags specify whether a newly bound member goes at the head or the tail of the list or completely replaces the list. To look up a name in a union directory, each member directory is searched in list order until the name is found. A bind flag specifies whether file creation is allowed in a member directory: a file created in the union directory goes in the first member directory in list order that allows creation, if any.

The glue that holds Plan 9 together is a network protocol called 9P, described in section 5 of this manual. All Plan 9 servers read and respond to 9P requests to navigate through a file tree and to perform operations such as reading and writing files within the tree.

Booting
When a terminal is powered on or reset, it must be told the name of a file server to boot from, the operating system kernel to boot, and a user name and password. How this dialog proceeds is environment- and machine-dependent. Once it is complete, the terminal loads a Plan 9 kernel, which sets some environment variables (see env(3)) and builds an initial name space. See namespace(4), boot(8), and init(8) for details, but some important aspects of the initial name space are:

- The environment variable $cputype is set to the name of the kernel’s CPU’s architecture: one of alpha, mips, sparc, power (Power PC), 386 (386, 486, Pentium, ...) etc. The environment variable $objctype is initially the same as $cputype.
- The environment variable $terminal is set to a description of the machine running the kernel, such as generic pc. Sometimes the middle word of $terminal encodes the file from which the kernel is booted; e.g. alpha apc axp is bootstrapped from /alpha/bapc.
- The environment variable $service is set to terminal. (Other ways of accessing Plan 9 may set $service to one of cpu, con, or rx.)
The environment variable $user is set to the name of the user who booted the terminal. The environment variable $home is set to that user's home directory.

/timeout/bin and /rc/bin are unioned into /bin.

After booting, the terminal runs the command interpreter, rc(1), on /usr/$user/lib/profile after moving to the user's home directory.

Here is a typical profile:

```
bind -a $home/bin/rc /bin
bind -a $home/bin/$cputype /bin
bind -c $home/tmp /tmp
font = /lib/font/bit/pelm/euro.9.font
upas/fs
switch($service){
case terminal
  plumber
  prompt=('term% ' ' ')
  exec rio -f $font
case cpu
  bind /mnt/term/dev/cons /dev/cons
  bind /mnt/term/dev/consctl /dev/consctl
  bind -a /mnt/term/mnt/wsys /dev
  prompt=('cpu% ' ' ')
  news
  case con
    prompt=('cpu% ' ' ')
  news
}
```

The first three lines replace /tmp with a tmp in the user's home directory and union personal bin directories with /bin, to be searched after the standard bin directories. The next starts the mail file system; see mail(1). Then different things happen, depending on the $service environment variable, such as running the window system rio(1) on a terminal.

To do heavy work such as compiling, the cpu(1) command connects a window to a CPU server; the same environment variables are set (to different values) and the same profile is run. The initial directory is the current directory in the terminal window where cpu was typed. The value of $service will be cpu, so the second arm of the profile switch is executed. The root of the terminal's name space is accessible through /mnt/term, so the bind is a way of making the window system's graphics interface (see draw(3)) available to programs running on the CPU server. The news(1) command reports current Plan 9 affairs.

The third possible service type, con, is set when the CPU server is called from a non-Plan-9 machine, such as through telnet (see con(1)).

### Using Plan 9

The user commands of Plan 9 are reminiscent of those in Research Unix, version 10. There are a number of differences, however.

The standard shell is rc(1), not the Bourne shell. The most noticeable differences appear only when programming and macro processing.

The character-delete character is backspace, and the line-kill character is control-U; these cannot be changed.

DEL is the interrupt character: typing it sends an interrupt to processes running in that window. See keyboard(6) for instructions on typing characters like DEL on the various keyboards.

If a program dies with something like an address error, it enters a 'Broken' state. It lingers, available for debugging with db(1) or acid(1). Broke (see kill(1)) cleans up broken processes.

The standard editor is one of acme(1) or sam(1). There is a variant of sam that permits running the file-manipulating part of sam on a non-Plan-9 system:

```
sam -r tcp!kremvax
```
For historical reasons, *sam* uses a tab stop setting of 8 spaces, while the other editors and window systems use 4 spaces. These defaults can be overridden by setting the value of the environment variable $TABSTOP to the desired number of spaces per tab.

Machine names may be prefixed by the network name, here tcp; and net for the system default.

Login connections and remote execution on non-Plan-9 machines are usually done by saying, for example,

```
con kremvax
```
or
```
rx deepthought chess
```
(see *con(1))*.

9fs connects to file systems of remote systems (see *srv(4))*. For example,

```
9fs kremvax
```
sets things up so that the root of kremvax’s file tree is visible locally in /n/kremvax.

*Faces(1)* gives graphical notification of arriving mail.

The Plan 9 file server has an integrated backup facility. The command

```
9fs dump
```
binds to /n/dump a tree containing the daily backups on the file server. The dump tree has years as top level file names, and month–day as next level file names. For example, /n/dump/2000/0120 is the root of the file system as it appeared at dump time on January 20, 2000. If more than one dump is taken on the same day, dumps after the first have an extra digit. To recover the version of this file as it was on June 15, 1999,

```
    cp /n/dump/1999/0615/sys/man/1/0intro.
```
or use *yesterday(1)*.

### SEE ALSO

This section for general publicly accessible commands.

Section (2) for library functions, including system calls.

Section (3) for kernel devices (accessed via *bind(1)*).

Section (4) for file services (accessed via *mount*).

Section (5) for the Plan 9 file protocol.

Section (6) for file formats.

Section (7) for databases and database access programs.

Section (8) for things related to administering Plan 9.

/sys/doc for copies of papers referenced in this manual.

The back of this volume has a permuted index to aid searches.

### DIAGNOSTICS

Upon termination each program returns a string called the *exit status*. It was either supplied by a call to *exits(2)* or was written to the command’s /proc/pid/ NOTE file (see *proc(3)*), causing an abnormal termination. The empty string is customary for successful execution; a non–empty string gives a clue to the failure of the command.
NAME
0a, 1a, 2a, 5a, 6a, 7a, 8a, ka, qa, va – assemblers

SYNOPSIS
2a [ option ... ] [ name ... ]
etc.

DESCRIPTION
These programs assemble the named files into object files for the corresponding architectures; see 2c(1) for the correspondence between an architecture and the character (1, 2, etc.) that specifies it. The assemblers handle the most common C preprocessor directives and the associated command-line options –D and –I. Other options are:

–o obj
 Place output in file obj (allowed only if there is just one input file). Default is to take the last element of the input path name, strip any trailing .s, and append .O, where O is first letter of the assembler’s name.

FILES
The directory /sys/include is searched for include files after machine-dependent files in /$objtype/include.

SOURCE
/sys/src/cmd/2a, etc.

SEE ALSO
2c(1), 2l(1).

Rob Pike, “A manual for the Plan 9 assembler”

BUGS
The list of assemblers given above is only partial, not all architectures are supported on all systems, some have been retired and some are provided by third parties.
NAME
0c, 1c, 2c, 5c, 6c, 7c, 8c, kc, qc, vc - C compilers

SYNOPSIS
2c [ option ... ] [ file ... ]
   etc.

DESCRIPTION
These commands compile the named C files into object files for the corresponding architecture. If there are multiple C files, the compilers will attempt to keep $NPROC compilations running concurrently. Associated with each compiler is a string objtype, for example

0c spim    little-endian MIPS 3000 family
1c 68000    Motorola MC68000
2c 68020    Motorola MC68020
5c arm      little-endian ARM
6c amd64    AMD64 and compatibles (e.g., Intel EM64T)
7c alpha    Digital Alpha APX
8c 386      Intel i386, i486, Pentium, etc.
kc sparc    Sun SPARC
qc power    Power PC
vc mips     big-endian MIPS 3000 family

The compilers handle most preprocessing directives themselves; a complete preprocessor is available in cpp(1), which must be run separately.

Let the first letter of the compiler name be O= 0, 1, 2, 5, 6, 7, 8, k, q, or v. The output object files end in .O. The letter is also the prefix of related programs: Oa is the assembler, O1 is the loader. Plan 9 conventionally sets the $objtype environment variable to the objtype string appropriate to the current machine's type. Plan 9 also conventionally has /objtype directories, which contain among other things: include, for machine-dependent include files; lib, for public object code libraries; bin, for public programs; and mkfile, for preconditioning mk(1).

The compiler options are:

-o obj   Place output in file obj (allowed only if there is just one input file). Default is to take the last element of the input file name, strip any trailing .c, and append .O.
-w       Print warning messages about unused variables, etc.
-B       Accept functions without a new-style ANSI C function prototype. By default, the compilers reject functions used without a defined prototype, although ANSI C permits them.
-Dname=def
-Dname   Define the name to the preprocessor, as if by #define. If no definition is given, the name is defined as 1.
-F       Enable type-checking of calls to print(2) and other formatted print routines. See the discussion of extensions, below.
-Idir    An #include file whose name does not begin with slash or is enclosed in double quotes is always sought first in the directory of the file argument. If this fails, the –. flag is given or the name is enclosed in <>, it is then sought in directories named in –I options, then in /sys/include, and finally in /$objtype/include.
- .       Suppress the automatic searching for include files in the directory of the file argument.
-N       Suppress automatic registerization and optimization.
-S       Print an assembly language version of the object code on standard output as well as generating the .O file.
−T Pass type signatures on all external and global entities. The signature is based on the C signo$E operator. See dynld(2).

−V By default, the compilers are non-standardly lax about type equality between void* values and other pointers; this flag requires ANSI C conformance.

−p Invoke a standard ANSI C preprocessor before compiling.

−a Instead of compiling, print on standard output acid functions (see acid(1)) for examining structures declared in the source files.

−aa Like −a except suppress information about structures declared in included header files.

−n When used with −a or −aa, places acid functions in file.acid for input file.c, and not on standard output.

The compilers support several extensions to ANSI C:

- A structure or union may contain unnamed substructures and subunions. The fields of the substructures or subunions can then be used as if they were members of the parent structure or union (the resolution of a name conflict is unspecified). When a pointer to the outer structure or union is used in a context that is only legal for the unnamed substructure, the compiler promotes the type and adjusts the pointer value to point at the substructure. If the unnamed structure or union is of a type with a tag name specified by a typedef statement, the unnamed structure or union can be explicitly referenced by <struct variable>..<tagname>.

- A structure value can be formed with an expression such as
  
  (struct S){v1, v2, v3}  

  where the list elements are values for the fields of struct S.

- Array initializers can specify the indices of the array in square brackets, as
  

  which initializes the third and tenth elements of the eleven-element array a.

- Structure initializers can specify the structure element by using the name following a period, as
  
  struct { int x; int y; } s = { .y 1, .x 5 };  

  which initializes elements y and then x of the structure s. These forms also accept the new ANSI C notation, which includes an equal sign:
  

  struct { int x; int y; } s = { .y = 1, .x = 5 };

- A global variable can be dedicated to a register by declaring it extern register in all modules and libraries.

- A #pragma of the form
  
  #pragma lib "libbio.a"  

  records that the program needs to be loaded with file/$objtype/lib/libbio.a: such lines, typically placed in library header files, obviate the −l option of the loaders. To help identify files in non-standard directories, within the file names in the #pragmas the string $M represents the name of the architecture (e.g., mips) and $O represents its identifying character (e.g., v).

- A #pragma of the form
  
  #pragma varargck argpos error 2  

  tells the compiler that the second argument to error is a print-like format string (see print(2)) that identifies the handling of subsequent arguments. The #pragma

  #pragma varargck type "s" char*  

  says that the format verb s processes an argument of type char*. The #pragma

  #pragma varargck flag 'c'  

  says that c is a flag character. These #pragmas are used, if the −F option is enabled, to type-check calls to print and other such routines.

- A #pragma with any of the following forms:
  
  #pragma incomplete type
  #pragma incomplete struct tag
  #pragma incomplete union tag

  where type is a typedef'd name for a structure or union type, and tag is a structure or union.
tag, tells the compiler that the corresponding type should have its signature calculated as an incomplete type even if it is subsequently fully defined. This allows the type signature mechanism to work in the presence of opaque types declared in header files, with their full definitions visible only to the code which manipulates them. With some imported software it might be necessary to turn off the signature generation completely for a large body of code (typically at the start and end of a particular include file). If \textit{type} is the word \_off\_, signature generation is turned off; if \textit{type} is the word \_on\_, the compiler will generate signatures.

- The C++ comment (// to end of line) is accepted as well as the normal convention of /* */.
- The compilers accept long long variables as a 64-bit type. The standard header typedefs this to \texttt{vlong}. Arithmetic on \texttt{vlong} values is usually emulated by a run-time library, though in at least 8c, only division and modulus use the run-time library and the other operators generate in-line code (and \texttt{uvlong-expression divison-or-modulus (1<<constant)} will turn into in-line bit operations, as is done for shorter \texttt{unsigned} expressions).

\textbf{EXAMPLE}

For the 68020, produce a program \texttt{prog} from C files \texttt{main.c} and \texttt{sub.c}:

\begin{verbatim}
  2c -FVw main.c sub.c
  2l -o prog main.2 sub.2
\end{verbatim}

\textbf{FILES}

\begin{verbatim}
/sys/include  system area for machine-independent \#include directives.
/sys/type/include  system area for machine-dependent \#include directives.
\end{verbatim}

\textbf{SOURCE}

\begin{verbatim}
/sys/src/cmd/cc  machine-independent part
/sys/src/cmd/2l, etc.  machine-dependent part
\end{verbatim}

\textbf{SEE ALSO}

2a(1), 2l(1), \texttt{cpp(1)}, \texttt{mk(1)}, \texttt{nm(1)}, \texttt{pcc(1)}, \texttt{db(1)}, \texttt{acid(1)}

Rob Pike, "How to Use the Plan 9 C Compiler"

\textbf{BUGS}

The list of compilers given above is only partial, not all architectures are supported on all systems, some have been retired and some are provided by third parties.

The default preprocessor only handles \#define, \#include, \#undef, \#ifdef, \#line, and \#ifndef. For a full ANSI preprocessor, use the \texttt{p} option.

The default search order for include files differs to that of \texttt{cpp(1)}.

Some features of C99, the 1999 ANSI C standard, are implemented.

\texttt{switch} expressions may not be either signedness of \texttt{vlong} on 32-bit architectures (8c at least).

The implementation of \texttt{vlong} assignment can use a static location and this can be disturbed by interrupts (e.g., notes) (8c at least).
NAME
   0l, 1l, 2l, 5l, 6l, 7l, 8l, kl, ql, vl – loaders

SYNOPSIS
   2l [ option ... ] [ file ... ]
   etc.

DESCRIPTION
   These commands load the named files into executable files for the corresponding architectures; see 2c(1) for the correspondence between an architecture and the character (1, 2, etc.) that specifies it. The files should be object files or libraries (archives of object files) for the appropriate architecture. Also, a name like -1ext represents the library libext.a in /$objtype/lib, where objtype is one of 68000, etc. as listed in 2c(1). The libraries must have tables of contents (see ar(1)).

   In practice, -1 options are rarely necessary as the header files for the libraries cause their archives to be included automatically in the load (see 2c(1)). For example, any program that includes header file libc.h causes the loader to search the C library /$objtype/lib/libc.a. Also, the loader creates an undefined symbol _main (or _mainp if profiling is enabled) to force loading of the startup linkage from the C library.

   The order of search to resolve undefined symbols is to load all files and libraries mentioned explicitly on the command line, and then to resolve remaining symbols by searching in topological order libraries mentioned in header files included by files already loaded. When scanning such libraries, the algorithm is to scan each library repeatedly until no new undefined symbols are picked up, then to start on the next library. Thus if library A needs B which needs A again, it may be necessary to mention A explicitly so it will be read a second time.

   The loader options are:
   -l       (As a bare option.) Suppress the default loading of the startup linkage and libraries specified by header files.
   -o out   Place output in file out. Default is O.out, where O is the first letter of the loader name.
   -p       Insert profiling code into the executable output; no special action is needed during compilation or assembly.
   -e       Insert (embedded) tracing code into the executable output; no special action is needed during compilation or assembly. The added code calls _tracein at function entries and _traceout at function exits.
   -s       Strip the symbol tables from the output file.
   -a       Print the object code in assembly language, with addresses.
   -v       Print debugging output that annotates the activities of the load.
   -M       (kl only) Generate instructions rather than calls to emulation routines for multiply and divide.
   -E symbol The entry point for the binary is symbol (default _main: _mainp under -p).
   -x [ file ] Produce an export table in the executable. The optional file restricts the exported symbols to those listed in the file. See dynld(2).
   -u [ file ] Produce an export table, import table and a dynamic load section in the executable. The optional file restricts the imported symbols to those listed in the file. See dynld(2).
   -t       (sl and vl only) Move strings into the text segment.
   -Hn      Executable header is type n. The meaning of the types is architecture-dependent; typically type 1 is Plan 9 boot format and type 2 is the regular Plan 9 format, the default. These are reversed on the MIPS. The Next boot format is 3. Type 4 in vl creates a MIPS executable for an SGI Unix system.
-Tt  The text segment starts at address t.
-Dd  The data segment starts at address d.
-Rr  The text segment is rounded to a multiple of r (if r is nonzero).

The numbers in the above options can begin with 0x or 0 to change the default base from decimal to hexadecimal or octal. The defaults for the values depend on the compiler and the header type.

The loaded image has several symbols inserted by the loader: etext is the address of the end of the text segment; bdata is the address of the beginning of the data segment; edata is the address of the end of the data segment; and end is the address of the end of the bss segment, and of the program.

FILES
   /$objtype/lib  for -l lib arguments.

SOURCE
   /sys/src/cmd/2l etc.

SEE ALSO
   2c(1), 2a(1), ar(1), nm(1), db(1), prof(1)
   Rob Pike, “How to Use the Plan 9 C Compiler”

BUGS
   The list of loaders given above is only partial, not all architectures are supported on all systems, some have been retired and some are provided by third parties.
NAME
abaco, readweb - browse the World-Wide Web

SYNOPSIS
abaco [-p] [-c ncols] [-m mtpt] [-t charset] [url]
readweb [url]

DESCRIPTION
Abaco is a lightweight web browser with the appearance of acme(1) with ncols columns (one by default). Given a url, it will start by displaying that page. Clicking mouse button 3 on a link opens it in a new abaco window. -t selects an alternate character set; -m an alternate mount point for webfs. Normally the standard error of subshells is closed, but -p prevents this.

Readweb imports the outside network, if necessary, starts webfs and webcookies and finally abaco.

FILES
/mnt/web   default webfs mount point

SOURCE
/sys/src/cmd/abaco
/rc/bin/readweb

SEE ALSO
vnc(1), webcookies(4), webfs(4),

BUGS
Abaco is a work in progress; many features of giant web browsers are absent.
NAME
cid, truss, trump - debugger

SYNOPSIS
acid [-kqw] [-l library] [-m machine] [ pid ] [ textfile ]
acid -l truss textfile
acid -l trump [ pid ] [ textfile ]

DESCRIPTION
Acid is a programmable symbolic debugger. It can inspect one or more processes that share an
address space. A program to be debugged may be specified by the process id of a running or
defunct process, or by the name of the program’s text file (8.out by default). At the prompt,
cid will store function definitions or print the value of expressions. Options are
-w Allow the textfile to be modified.
-q Print variable renamings at startup.
-l library Load from library at startup; see below.
-m machine Assume instructions are for the given CPU type (one of alpha, 386, etc., as listed
in 2c(1), or sunsparc or mipsco for the manufacturer-defined instruction notation for those processors) instead of using the magic number to select the CPU type.
-k Debug the kernel state for the process, rather than the user state.

At startup, acid obtains standard function definitions from the library file
/sys/lib/acid/port, architecture-dependent functions from
/sys/lib/acid/$objtype, user-specified functions from $home/lib/acid, and further
functions from -l files. Definitions in any file may override previously defined functions. If the
function acidinit() is defined, it will be invoked after all libraries have been loaded. See 2c(1) for
information about creating acid functions for examining data structures.

Language
Symbols of the program being debugged become integer variables whose values are addresses.
Contents of addresses are obtained by indirection. Local variables are qualified by function name,
for example main:argv. When program symbols conflict with acid words, distinguishing $ signs are prefixed. Such renamings are reported at startup if the option –q is enabled.

Variable types (integer, float, list, string) and formats are inferred from assignments. Truth values
false/true are attributed to zero/nonzero integers or floats and to empty/nonempty lists or
strings. Lists are sequences of expressions surrounded by {} and separated by commas.

Expressions are much as in C, but yield both a value and a format. Casts to complex types are
allowed. Lists admit the following operators, with subscripts counted from 0.

head list
tail list
append list, element
delete list, subscript

Format codes are the same as in db(1). Formats may be attached to (unary) expressions with \,
e.g. (32*7)\D. There are two indirection operators, * to address a core image, @ to address a
text file. The type and format of the result are determined by the format of the operand, whose
type must be integer.

Statements are
if expr then statement [ else statement ]
while expr do statement
loop expr, expr do statement
defn name( args ) { statement }
defn name
name( args )
builtin name( args )
local name
return expr
   whatis [ name]

The statement defn name clears the definition for name. A defn may override a built-in function; prefixing a function call with builtin ignores any overriding defn, forcing the use of the built-in function.

Here is a partial list of functions; see the manual for a complete list.

stk() Print a stack trace for current process.
lstk() Print a stack trace with values of local variables.
gpr() Print general registers. Registers can also be accessed by name, for example *R0.
spr() Print special registers such as program counter and stack pointer.
fpr() Print floating-point registers.
regs() Same as spr();gpr().
fmt(expr, format) Expression expr with format given by the character value of expression format.
src(address) Print 10 lines of source around the program address.
src(address) Get the source line for the program address into a window of a running sam(1) and select it.
line(address) Print source line nearest to the program address.
source() List current source directories.
addsrcdir(string) Add a source directory to the list.
filepc(where) Convert a string of the form sourcefile:linenumber to a machine address.
pfile(address) Convert a machine address to a source file name.
pcline(address) Convert a machine address to a source line number.
bptab() List breakpoints set in the current process.
bpset(address) Set a breakpoint in the current process at the given address.
bpdel(address) Delete a breakpoint from the current process.
cont() Continue execution of current process and wait for it to stop.
step() Execute a single machine instruction in the current process.
func() Step repeatedly until after a function return.
stopped(pid) This replaceable function is called automatically when the given process stops. It normally prints the program counter and returns to the prompt.
asm(address) Disassemble 30 machine instructions beginning at the given address.
mem(address, string) Print a block of memory interpreted according to a string of format codes.
dump(address, n, string) Like mem(), repeated for n consecutive blocks.
print(expr,...) Print the values of the expressions.
newproc(arguments) Start a new process with arguments given as a string and halt at the first instruction.
new() Like newproc(), but take arguments (except argv[0]) from string variable progargs.
win() Like new(), but run the process in a separate window.
start(pid) Start a stopped process.
kill(pid) Kill the given process.
setproc(pid) Make the given process current.
rc(string) Escape to the shell, rc(1), to execute the command string.

Libraries

There are a number of acid 'libraries' that provide higher-level debugging facilities. Two notable examples are truss and trump, which use acid to trace system calls (truss) and memory allocation (trump). Both require starting acid on the program, either by attaching to a running process or by executing new() on a binary (perhaps after setting progargs), stopping the process, and then running truss() or trump() to execute the program under the scaffolding. The output will be a trace of the system calls (truss) or memory allocation and free calls (trump) executed by the program. When finished tracing, stop the process and execute untruss() or untrump()
EXAMPLES

Start to debug /bin/ls; set some breakpoints; run up to the first one:

% acid /bin/ls
/bin/ls: mips plan 9 executable
/sys/lib/acid/port
/sys/lib/acid/mips
acid: new()
70094: system call _main ADD $−0x14,R29
70094: breakpoint main+0x4 MOVW R31,0x0(R29)
acid: pid
70094
acid: argv0 = **main:argv\s
acid: whatis argv0
integer variable format s
acid: *argv0
/bin/ls
acid: bpset(ls)
acid: cont()
70094: breakpoint ls ADD $−0x16c8,R29
acid:

Display elements of a linked list of structures:

complex Str { 'D' 0 val; 'X' 4 next; };
complex Str s;
s = *headstr;
while s != 0 do{
    print(s.val, "\n");
    s = s.next;
}

Note the use of the . operator instead of ->.

Display an array of bytes declared in C as char array[].

*(array\s)

This example gives array string format, then prints the string beginning at the address (in acid notation) *array.

Trace the system calls executed by ls():

% acid −l truss /bin/ls
/bin/ls:386 plan 9 executable
/sys/lib/acid/port
/sys/lib/acid/kernel
/sys/lib/acid/truss
/sys/lib/acid/386
acid: progargs = "−l lib/profile"
acid: new()
acid: truss()
open("#c/pid", 0)
    return value: 3
pread(3, 0x7fffeeac, 20, −1)
    return value: 12
data: " 166 
...
stat("lib/profile", 0x0000f8cc, 113)
    return value: 65
open("/env/timezone", 0)
    return value: 3
pread(3, 0x7fffd7c4, 1680, -1)
    return value: 1518
    data: "EST −18000 EDT −14400
9943200 25664400 41392800 57718800 73447200 89168400
104896800 ...
" 
close(3)
    return value: 0
pwrite(1, "--rw−rw−r-- M 9 rob rob 2519 Mar 22 10:29 lib/profile ", 54, -1)
    --rw−rw−r-- M 9 rob rob 2519 Mar 22 10:29 lib/profile
    return value: 54
...
166: breakpoint _exits+0x5 INTB $0x40
acid: cont()

FILES
    /proc/*/text
    /proc/*/mem
    /proc/*/ctl
    /proc/*/note
    /sys/lib/acid/$objtype
    /sys/lib/acid/port
    /sys/lib/acid/kernel
    /sys/lib/acid/trump
    /sys/lib/acid/truss
    $home/lib/acid

SOURCE
    /sys/src/cmd/acid

SEE ALSO
    2a(1), 2c(1), 2l(1), mk(1), db(1)
    Phil Winterbottom, "Acid Manual".

DIAGNOSTICS
    At termination, kill commands are proposed for processes that are still active.

BUGS
    There is no way to redirect the standard input and standard output of a new process.
    Source line selection near the beginning of a file may pick an adjacent file.
    With the extant stepping commands, one cannot step through instructions outside the text segment and it is hard to debug across process forks.
NAME
acme, win, awd – interactive text windows

SYNOPSIS
acme [-ab][−c ncol][−F varfont][−F fixfont][−l loadfile | file ...]
win[ command]
awd[ label]

DESCRIPTION
Acme manages windows of text that may be edited interactively or by external programs. The interactive interface uses the keyboard and mouse; external programs use a set of files served by acme; these are discussed in acme(4).

Any named files are read into acme windows before acme accepts input. With the −l option, the state of the entire system is loaded from loadfile, which should have been created by a Dump command (q.v.), and subsequent file names are ignored. Plain files display as text; directories display as columnated lists of the names of their components, as in ls −p directory|mc except that the names of subdirectories have a slash appended.

The −F (−F) option sets the main font, usually variable−pitch (alternate, usually fixed−pitch); the default is /lib/font/bit/lucidasans/euro.8.font (..../lucm/unicode.9.font). Tab intervals are set to the width of 4 (or the value of $tabstop) numeral zeros in the appropriate font.

Windows
Acme windows are in two parts: a one−line tag above a multi−line body. The body typically contains an image of a file, as in sam(1), or the output of a program, as in an rio(1) window. The tag contains a number of blank−separated words, followed by a vertical bar character, followed by anything. The first word is the name of the window, typically the name of the associated file or directory, and the other words are commands available in that window. Any text may be added after the bar; examples are strings to search for or commands to execute in that window. Changes to the text left of the bar will be ignored, unless the result is to change the name of the window.

If a window holds a directory, the name (first word of the tag) will end with a slash.

Scrolling
Each window has a scroll bar to the left of the body. The scroll bar behaves much as in sam(1) or rio(1) except that scrolling occurs when the button is pressed, rather than released, and continues as long as the mouse button is held down in the scroll bar. For example, to scroll slowly through a file, hold button 3 down near the top of the scroll bar. Moving the mouse down the scroll bar speeds up the rate of scrolling.

Layout
Acme windows are arranged in columns. By default, it creates two columns when starting; this can be overridden with the −c option. Placement is automatic but may be adjusted using the layout box in the upper left corner of each window and column. Pressing and holding any mouse button in the box drags the associated window or column. For windows, just clicking in the layout box grows the window in place: button 1 grows it a little, button 2 grows it as much as it can, still leaving all other tags in that column visible, and button 3 takes over the column completely, temporarily hiding other windows in the column. (They will return en masse if any of them needs attention.) The layout box in a window is normally white; when it is black in the center, it records that the file is ‘dirty’: acme believes it is modified from its original contents.

Tags exist at the top of each column and across the whole display. Acme pre−loads them with useful commands. Also, the tag across the top maintains a list of executing long−running commands.

Typing
The behavior of typed text is similar to that in rio(1) except that the characters are delivered to the tag or body under the mouse; there is no ‘click to type’. (The experimental option −b causes typing to go to the most recently clicked−at or made window.) The usual backspacing conventions apply. As in sam(1) but not rio, the ESC key selects the text typed since the last mouse action, a feature particularly useful when executing commands. A side effect is that typing ESC with text already selected is identical to a Cut command (q.v.).
Most text, including the names of windows, may be edited uniformly. The only exception is that the command names to the left of the bar in a tag are maintained automatically; changes to them are repaired by acme.

When a window is in autoindent mode (see the Indent command below) and a newline character is typed, acme copies leading white space on the current line to the new line. The option -a causes each window to start in autoindent mode.

Directory context

Each window’s tag names a directory: explicitly if the window holds a directory; implicitly if it holds a regular file (e.g. the directory /adm if the window holds /adm/users). This directory provides a context for interpreting file names in that window. For example, the string users in a window labeled /adm/ or /adm/keys will be interpreted as the file name /adm/users. The directory is defined purely textually, so it can be a non-existent directory or a real directory associated with a non-existent file (e.g. /adm/not-a-file). File names beginning with a slash are assumed to be absolute file names.

Errors

Windows whose names begin with - or + conventionally hold diagnostics and other data not directly associated with files. A window labeled +Errors receives all diagnostics produced by acme itself. Diagnostics from commands run by acme appear in a window named directory/+Errors where directory is identified by the context of the command. These error windows are created when needed.

Mouse button 1

Mouse button 1 selects text just as in sam(1) or rio(1), including the usual double-clicking conventions.

Mouse button 2

By an action similar to selecting text with button 1, button 2 indicates text to execute as a command. If the indicated text has multiple white-space-separated words, the first is the command name and the second and subsequent are its arguments. If button 2 is ‘clicked’—indicates a null string—acme expands the indicated text to find a command to run: if the click is within button-1—selected text, acme takes that selection as the command; otherwise it takes the largest string of valid file name characters containing the click. Valid file name characters are alphanumerics and . - + / . This behavior is similar to double-clicking with button 1 but, because a null command is meaningless, only a single click is required.

Some commands, all by convention starting with a capital letter, are built-ins that are executed directly by acme:

Cut Delete most recently selected text and place in snarf buffer.

Del Delete window. If window is dirty, instead print a warning; a second Del will succeed.

Delcol Delete column and all its windows, after checking that windows are not dirty.

Delete Delete window without checking for dirtiness.

Dump Write the state of acme to the file name, if specified, or $home/acme.dump by default.

Edit Treat the argument as a text editing command in the style of sam(1). The full Sam language is implemented except for the commands k, n, q, and !. The = command is slightly different: it includes the file name and gives only the line address unless the command is explicitly =#. The ‘current window’ for the command is the body of the window in which the Edit command is executed. Usually the Edit command would be typed in a tag; longer commands may be prepared in a scratch window and executed, with Edit itself in the current window, using the 2-1 chord described below.

Exit Exit acme after checking that windows are not dirty.

Font With no arguments, change the font of the associated window from fixed-spaced to proportional-spaced or vice versa. Given a file name argument, change the font of the window to that stored in the named file. If the file name argument is prefixed by var(fix), also set the default proportional-spaced (fixed-spaced) font for future use to that font. Other existing windows are unaffected.
Get Load file into window, replacing previous contents (after checking for dirtiness as in Del). With no argument, use the existing file name of the window. Given an argument, use that file but do not change the window’s file name.

ID Print window ID number (q.v.).

Incl When opening ‘include’ files (those enclosed in <>) with button 3, acme searches in directories /$objtype/include and /sys/include. Incl adds its arguments to a supplementary list of include directories, analogous to the –I option to the compilers. This list is per-window and is inherited when windows are created by actions in that window, so Incl is most usefully applied to a directory containing relevant source. With no arguments, Incl prints the supplementary list. This command is largely superseded by plumbing (see *plumb*(6)).

Indent Set the autoindent mode according to the argument: on and off set the mode for the current window; ON and OFF set the mode for all existing and future windows.

Kill Send a kill note to acme-initiated commands named as arguments.

Load Restore the state of acme from a file (default $home/acme.dump) created by the Dump command.

Local When prefixed to a command run the command in the same file name space and environment variable group as acme. The environment of the command is restricted but is sufficient to run bind(1), 9fs (see srv(4)), import(4), etc., and to set environment variables such as $objtype.

Look Search in body for occurrence of literal text indicated by the argument or, if none is given, by the selected text in the body.

New Make new window. With arguments, load the named files into windows.

Newcol Make new column.

Paste Replace most recently selected text with contents of snarf buffer.

Put Write window to the named file. With no argument, write to the file named in the tag of the window.

Putall Write all dirty windows whose names indicate existing regular files.

Redo Complement of Undo.

Send Append selected text or snarf buffer to end of body; used mainly with *win*.

Snarf Place selected text in snarf buffer.

Sort Arrange the windows in the column from top to bottom in lexicographical order based on their names.

Tab Set the width of tab stops for this window to the value of the argument, in units of widths of the zero character. With no arguments, it prints the current value.

Undo Undo last textual change or set of changes.

Zerox Create a copy of the window containing most recently selected text.

<|> If a regular shell command is preceded by a <, |, or > character, the selected text in the body of the window is affected by the I/O from the command. The < character causes the selection to be replaced by the standard output of the command; > causes the selection to be sent as standard input to the command; and | does both at once, ‘piping’ the selection through the command and replacing it with the output.

A common place to store text for commands is in the tag; in fact acme maintains a set of commands appropriate to the state of the window to the left of the bar in the tag.
If the text indicated with button 2 is not a recognized built-in, it is executed as a shell command. For example, indicating date with button 2 runs date(1). The standard and error outputs of commands are sent to the error window associated with the directory from which the command was run, which will be created if necessary. For example, in a window /adm/users executing pwd will produce the output /adm in a (possibly newly-created) window labeled /adm/+Errors; in a window containing /sys/src/cmd/sam/sam/sam.c executing mk will run mk(1) in /sys/src/cmd/sam, producing output in a window labeled /sys/src/cmd/sam/+Errors. The environment of such commands contains the variable % with value set to the filename of the window in which the command is run, and $winid set to the window’s id number (see acme(4)).

Mouse button 3

Pointing at text with button 3 instructs acme to locate or acquire the file, string, etc. described by the indicated text and its context. This description follows the actions taken when button 3 is released after sweeping out some text. In the description, text refers to the text of the original sweep or, if it was null, the result of applying the same expansion rules that apply to button 2 actions.

If the text names an existing window, acme moves the mouse cursor to the selected text in the body of that window. If the text names an existing file with no associated window, acme loads the file into a new window and moves the mouse there. If the text is a file name contained in angle brackets, acme loads the indicated include file from the directory appropriate to the suffix of the file name of the window holding the text. (The Incl command adds directories to the standard list.)

If the text begins with a colon, it is taken to be an address, in the style of sam(1), within the body of the window containing the text. The address is evaluated, the resulting text highlighted, and the mouse moved to it. Thus, in acme, one must type :/regexp or :127 not just regexp or 127. (There is an easier way to locate literal text; see below.)

If the text is a file name followed by a colon and an address, acme loads the file and evaluates the address. For example, clicking button 3 anywhere in the text file.c:27 will open file.c, select line 27, and put the mouse at the beginning of the line. The rules about Error files, directories, and so on all combine to make this an efficient way to investigate errors from compilers, etc.

If the text is not an address or file, it is taken to be literal text, which is then searched for in the body of the window in which button 3 was clicked. If a match is found, it is selected and the mouse is moved there. Thus, for search occurrences of a word in a file, just click button 3 on the word. Because of the rule of using the selection as the button 3 action, subsequent clicks will find subsequent occurrences without moving the mouse.

In all these actions, the mouse motion is not done if the text is a null string within a non-null selected string in the tag, so that (for example) complex regular expressions may be selected and applied repeatedly to the body by just clicking button 3 over them.

Chords of mouse buttons

Several operations are bound to multiple-button actions. After selecting text, with button 1 still down, pressing button 2 executes Cut and button 3 executes Paste. After clicking one button, the other undoes the first; thus (while holding down button 1) 2 followed by 3 is a Snarf that leaves the file undirtied; 3 followed by 2 is a no-op. These actions also apply to text selected by double-clicking because the double-click expansion is made when the second click starts, not when it ends.

Commands may be given extra arguments by a mouse chord with buttons 2 and 1. While holding down button 2 on text to be executed as a command, clicking button 1 appends the text last pointed to by button 1 as a distinct final argument. For example, to search for literal text one may execute Look text with button 2 or instead point at text with button 1 in any window, release button 1, then execute Look, clicking button 1 while 2 is held down.

When an external command (e.g. echo(1)) is executed this way, the extra argument is passed as expected and an environment variable $acmeaddr is created that holds, in the form interpreted by button 3, the fully-qualified address of the extra argument.

Support programs

Win creates a new acme window and runs a command (default /bin/rc) in it, turning the window
into something analogous to an *rio*(1) window. Executing text in a *win* window with button 2 is similar to using Send.

*awd* loads the tag line of its window with the directory in which it's running, suffixed *–label* (default *rc*); it is intended to be executed by a cd function for use in *win* windows. An example definition is

```
fn cd { builtin cd $1 && awd $sysname }
```

### Applications and guide files

In the directory *acme* live several subdirectories, each corresponding to a program or set of related programs that employ *acme*’s user interface. Each subdirectory includes source, binaries, and a readme file for further information. It also includes a guide, a text file holding sample commands to invoke the programs. The idea is to find an example in the guide that best matches the job at hand, edit it to suit, and execute it.

Whenever a command is executed by *acme*, the default search path includes the directory of the window containing the command and its subdirectory *$cputype*. The program directories in *acme* contain appropriately labeled subdirectories of binaries, so commands named in the guide files will be found automatically when run. Also, *acme* binds the directories *acme/bin* and *acme/bin/$cputype* to the beginning of *bin* when it starts; this is where *acme*-specific programs such as *win* and *awd* reside.

#### FILES

- **$home/acme.dump** default file for Dump and Load; also where state is written if *acme* dies or is killed unexpectedly, e.g. by deleting its window.
- **/acme/*/guide** template files for applications
- **/acme/*/readme** informal documentation for applications
- **/acme/*/src** source for applications
- **/acme/*/mips** MIPS-specific binaries for applications

#### SOURCE

- **/sys/src/cmd/acme**
- **/acme/bin/source/win**
- **/sys/src/cmd/awd.c**

#### SEE ALSO

- *acme*(4)
- Rob Pike, *Acme: A User Interface for Programmers*.

#### BUGS

With the *–l* option or Load command, the recreation of windows under control of external programs such as *win* is just to rerun the command; information may be lost.
NAME
ansitize - translate Plan 9 C to ANSI C

SYNOPSIS
ansitize [-c conf] [-I dir]... [-p preload]... [file]

DESCRIPTION
Ansitize translates programs written in the Plan 9 C dialect into standard ANSI C programs, preserving comments and formatting.

The options are:
- `-c conf`
  Read configuration information from the file `conf`. The format of the configuration file is discussed below.
- `-I dir` Add `dir` to the list of directories searched for `#include` files. `/386/include` and `/sys/include` are added to the list after processing the `-I` options.
- `-p preload`
  Before processing `file`, process the file `preload`, but do not print its translation. This option is useful mainly for translating header files. See the examples below.

Ansitize translates many constructs from Plan 9 C, described below. It does not translate types or other features present in the Plan 9 C environment when those features can be provided by appropriate program context. For example, ansitize removes long character constants and strings but still assumes that Rune is a defined type.

Ansitize translates the following constructs.

anonymous structures or unions
Plan 9 C allows anonymous structures and unions. Ansitize gives these explicit names and translates references to reflect the new names. If a struct (or union) name is declared anonymously, ansitize uses `_name` in the new declaration. Otherwise, unions are named `u`, `u2`, etc., and structures are named `_1`, `_2`, etc. For example, by default ansitize translates the first structure definition into the second:

```c
struct A {  
  union {
    int x;
    int y;
  };
  struct B;  
};
```

These default names can be overridden by a configuration line `rename old new`, where `old` is a single `name` or is `tag.name`, which restricts the renaming to the elements of struct (or union) `tag`. For example, using a configuration:

```
rename A.u au
rename _B b
(or rename A._b b)
```

would produce the third structure definition above.

anonymous structure promotions
Plan 9 C allows pointers to structures with anonymous elements to be passed to functions expecting pointers to the anonymous elements. For example, given the structure definition above, if a `struct A *a` is passed to a function expecting a `struct B *`, the C compiler instead passes a pointer to the `B` inside the `A`. Ansitize does the same transformation, in this case rewriting `f(a)` to `f(&a->b)`. The same conversion applies to simple assignment of `struct A *` to `struct B *`.

anonymous function parameters
Plan 9 C does not require unused function parameters to be named in the function definition. Ansitize names these parameters `_1`, `_2`, etc. For example, ansitize rewrites

```c
Rune struct A *a
```
void main(int, char**) { }

into

void main(int _1, char** _2) { }

structure displays
Plan 9 C allows casted initializer lists as structure values, as in (Point){1,2}. Ansitize can rewrite these into function calls, as in pt(1,2), but only does so if directed by a configuration line reconstruct struct-name function-name, as in reconstruct Point pt.

Unicode identifiers
Ansitize rewrites identifiers containing Unicode characters into ASCII equivalents, replacing Greek letters with their names and other Unicode characters with _xxxx, where xxxx is the hexadecimal value of the character.

long character constants
Ansitize rewrites long character constants like L'\n', L'a', or L'ý' into equivalent expressions like '\n', 'a', or (Rune)0x00FF.

long string constants
Ansitize replaces Rune string constants like L"abc" with references to statically declared arrays with names derived from the string data. It recognizes the special case where a Rune string is being used to initialize a Rune array and replaces the string in that case with an array. For example, ansitize rewrites the first program into the second:

Rune L_abcs[] = {'a','b','c',0};
Rune *x = L"abc";
Rune y[] = L"def";

#pragma lines
Ansitize places #pragma lines inside /* */ comments. #pragma varargck lines are handled separately and are placed inside #ifdef VARARGCK / #endif pairs. (At least one compiler under development for Unix recognizes these #pragmas.)

integer/pointer casts
Some overeager Unix compilers complain about casts from integer to pointer, even when the pointer is as wide as or wider than the integer. Ansitize inserts an extra (uintptr) cast to silence these warnings: p=(void*)i becomes p=(void*)(uintptr)i.

<ctypes.h> casts
The macros defined in Plan 9's <ctypes.h> cast their arguments to uchar so that either signed or unsigned character arguments can be passed to them. Unix's <ctypes.h> requires the use of unsigned character arguments. Ansitize adds casts as necessary to the arguments of isalpha, isdigit, toupper, etc.

EXAMPLES
A configuration file for translating the regexp(2) library:

rename Resub.uy s
rename Resub.uly e
rename Reinst.uy ul
rename Reinst.uly u2

Translate the source files:

cd /sys/src/libregexp
for(i in *.c)
    ansitize -c conf $i >$i.ansi

Translate the header file, reading <u.h> and <libc.h> first for context:

cd /sys/include
ansitize -p /386/include/u.h -p libc.h regexp.h >regexp.h.ansi

SOURCE
/sys/src/cmd/ansitize

SEE ALSO
2c(1), fortune(1)
Rob Pike, “How to use the Plan 9 C Compiler”

**BUGS**

*Ansitize* stops short of full checking of the input program. Test that they compile using `2c(1)` before running *ansitize*.

*Ansitize* ignores `#ifdef` and `#define`, limiting the kinds of macros that can be used. In particular, macros that introduce new control flow constructs will confuse the parser. (The parser contains extra grammar productions to accommodate the `arg(2)` macros and `va_arg`.)
NAME
   ap - fetch Associated Press news articles

SYNOPSIS
   ap [article-name]

DESCRIPTION
   ap fetches Associated Press news articles from http://www.newsday.com. Without any arguments it provides a two column list of article keys and descriptions. When invoked with an article key it fetches that article.

SOURCE
   /rc/bin/ap
NAME
  ar - archive and library maintainer
SYNOPSIS
  ar key [ posname ] afile [ file ...]
DESCRIPTION
  Ar maintains groups of files combined into a single archive file, afile. The main use of ar is to create and update library files for the loaders 2(l), etc. It can be used, though, for any similar purpose.
  Key is one character from the set drqtmpx, optionally concatenated with one or more of vuaibclo. The files are constituents of the archive afile. The meanings of the key characters are:
  d  Delete files from the archive file.
  r  Replace files in the archive file, or add them if missing. Optional modifiers are
     u  Replace only files with modified dates later than that of the archive.
     a  Place new files after posname in the archive rather than at the end.
     b or i Place new files before posname in the archive.
  q  Quick. Append files to the end of the archive without checking for duplicates. Avoids quadratic behavior in for (i in *.v) ar r lib.a $i.
  t  List a table of contents of the archive. If names are given, only those files are listed.
  p  Print the named files in the archive.
  m  Move the named files to the end or elsewhere, specified as with r.
  o  Preserve the access and modification times of files extracted with the x command.
  x  Extract the named files. If no names are given, all files in the archive are extracted. In neither case does x alter the archive file.
  v  Verbose. Give a file-by-file description of the making of a new archive file from the old archive and the constituent files. With p, precede each file with a name. With t, give a long listing of all information about the files, somewhat like a listing by ls(l), showing
     mode uid/gid size date name
  l  Local. Normally ar places its temporary files in the directory /tmp. This option causes them to be placed in the local directory.
  When a d, r, or m key is specified and all members of the archive are valid object files for the same architecture, ar inserts a table of contents, required by the loaders, at the front of the library. The table of contents is rebuilt whenever the archive is modified, except when the q key is specified or when the table of contents is explicitly moved or deleted.
EXAMPLE
  ar cr lib.a *.v
     Replace the contents of library lib.a with the object files in the current directory.
FILES
  /tmp/v*        temporaries
SOURCE
  /sys/src/cmd/ar.c
SEE ALSO
  2(l), ar(6)
BUGS
  If the same file is mentioned twice in an argument list, it may be put in the archive twice.
  This command predates Plan 9 and makes some invalid assumptions, for instance that user id's are numeric.
NAME
ascii, unicode - interpret ASCII, Unicode characters

SYNOPSIS
ascii [-8cnt][-dox | -b n][ text]
unicode hexmin–hexmax
unicode [-t] hex[... ]
unicode [-n] characters
look hex /lib/unicode

DESCRIPTION
Ascii prints the ASCII values corresponding to characters and vice versa; under the -8 option, the
ISO Latin-1 extensions (codes 0200–0377) are included. The values are interpreted in a settable
numeric base; -o specifies octal, -d decimal, -x hexadecimal (the default), and -b n base n.

With no arguments, ascii prints a table of the character set in the specified base. Characters of
text are converted to their ASCII values, one per line. If, however, the first text argument is a valid
number in the specified base, conversion goes the opposite way. Control characters are printed as
two- or three-character mnemonics. Other options are:
-n  Force numeric output.
-c  Force character output.
-t  Convert from numbers to running text; do not interpret control characters or insert new-
lines.

Unicode is similar; it converts between UTF and character values from the Unicode Standard (see
utf(6)). If given a range of hexadecimal numbers, unicode prints a table of the specified Unicode
characters — their values and UTF representations. Otherwise it translates from UTF to numeric
value or vice versa, depending on the appearance of the supplied text; the -n option forces
numeric output to avoid ambiguity with numeric characters. If converting to UTF , the characters
are printed one per line unless the -t flag is set, in which case the output is a single string con-
taining only the specified characters. Unlike ascii, unicode treats no characters specially.

The output of ascii and unicode may be unhelpful if the characters printed are not available in the
current font.

The file /lib/unicode contains a table of characters and descriptions, sorted in hexadecimal
order, suitable for look(1) on the lower case hex values of characters.

EXAMPLES
ascii  -d
      Print the ASCII table base 10.

      unicode  p
      Print the hex value of 'p'.

      unicode  2200–22f1
      Print a table of miscellaneous mathematical symbols.

      look  039 /lib/unicode
      See the start of the Greek alphabet's encoding in the Unicode Standard.

FILES
/lib/unicode table of characters and descriptions.

SOURCE
/sys/src/cmd/ascii.c
/sys/src/cmd/unicode.c

SEE ALSO
look(1), tcs(1), utf(6), font(6)
NAME
awk - pattern-directed scanning and processing language

SYNOPSIS
awk [ -f file ] [-d d] [-mf n | -mr n] [-safe] [-v var=value] [ -f profile ] [ prog ] [ file ... ]

DESCRIPTION
Awk scans each input file for lines that match any of a set of patterns specified literally in prog or in one or more files specified as -f profile. With each pattern there can be an associated action that will be performed when a line of a file matches the pattern. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. The file name - means the standard input. Any file of the form var=value is treated as an assignment, not a file name, and is executed at the time it would have been opened if it were a file name. The option -v followed by var=value is an assignment to be done before the program is executed; any number of -v options may be present. -F fs option defines the input field separator to be the regular expression fs.

An input line is normally made up of fields separated by white space, or by regular expression FS. The fields are denoted $1, $2, ..., while $0 refers to the entire line. If FS is null, the input line is split into one field per character.

To compensate for inadequate implementation of storage management, the -mr option can be used to set the maximum size of the input record, and the -mf option to set the maximum number of fields.

The -safe option causes awk to run in "safe mode," in which it is not allowed to run shell commands or open files and the environment is not made available in the ENVIRON variable.

A pattern-action statement has the form

pattern { action }

A missing { action } means print the line; a missing pattern always matches. Pattern-action statements are separated by newlines or semicolons.

An action is a sequence of statements. A statement can be one of the following:

if( expression ) statement [ else statement ]
while( expression ) statement
for( expression ; expression ; expression ) statement
for( var in array ) statement
do statement while( expression )
break
continue
{ [ statement ... ] }  
expression # commonly var = expression
print [ expression-list ] [ > expression ]
printf format [, expression-list ] [ > expression ]
return [ expression ]
next     # skip remaining patterns on this input line
nextfile # skip rest of this file, open next, start at top
delete array[ expression ] # delete an array element
delete array # delete all elements of array
exit [ expression ] # exit immediately; status is expression

Statements are terminated by semicolons, newlines or right braces. An empty expression-list stands for $0. String constants are quoted " ", with the usual C escapes recognized within. Expressions take on string or numeric values as appropriate, and are built using the operators + - * / % ^ (exponentiation), and concatenation (indicated by white space). The operators ! ++ -- += -= *= /= %= ^= >= <= == != ?: are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. Multiple subscripts such as [ i, j, k ] are permitted; the constituents are concatenated, separated by the value of SUBSEP.
The print statement prints its arguments on the standard output (or on a file if > file or >> file is present or on a pipe if | cmd is present), separated by the current output field separator, and terminated by the output record separator. file and cmd may be literal names or parenthesized expressions; identical string values in different statements denote the same open file. The printf statement formats its expression list according to the format (see fprintf(2)). The built-in function close(expr) closes the file or pipe expr. The built-in function flush(expr) flushes any buffered output for the file or pipe expr. If expr is omitted or is a null string, all open files are flushed.

The mathematical functions exp, log, sqrt, sin, cos, and atan2 are built in. Other built-in functions:

- **length** if its argument is a string, the string's length is returned. If its argument is an array, the number of subscripts in the array is returned. If no argument, the length of $0$ is returned.
- **rand** random number on (0,1)
- **srand** sets seed for rand and returns the previous seed.
- **int** truncates to an integer value
- **utf** converts its numerical argument, a character number, to a UTF string
- **substr(s, m, n)** the $n$–character substring of $s$ that begins at position $m$ counted from 1.
- **index(s, t)** the position in $s$ where the string $t$ occurs, or 0 if it does not.
- **match(s, r)** the position in $s$ where the regular expression $r$ occurs, or 0 if it does not. The variables RSTART and RLENGTH are set to the position and length of the matched string.
- **split(s, a, fs)** splits the string $s$ into array elements $a[1]$, $a[2]$, ..., $a[n]$, and returns $n$. The separation is done with the regular expression $fs$ or with the field separator $FS$ if $fs$ is not given. An empty string as field separator splits the string into one array element per character.
- **sub(r, t, s)** substitutes $t$ for the first occurrence of the regular expression $r$ in the string $s$. If $s$ is not given, $0$ is used.
- **gsub** same as sub except that all occurrences of the regular expression are replaced; sub and gsub return the number of replacements.
- **sprintf(fmt, expr, ...)** the string resulting from formatting expr ... according to the printf format fmt
- **system(cmd)** executes cmd and returns its exit status
- **tolower(str)** returns a copy of str with all upper–case characters translated to their corresponding lower–case equivalents.
- **toupper(str)** returns a copy of str with all lower–case characters translated to their corresponding upper–case equivalents.

The “function” getline sets $0$ to the next input record from the current input file; getline < file sets $0$ to the next record from file. getline x sets variable $x$ instead. Finally, cmd | getline pipes the output of cmd into getline; each call of getline returns the next line of output from cmd. In all cases, getline returns 1 for a successful input, 0 for end of file, and -1 for an error.

Patterns are arbitrary Boolean combinations (with ! | || & &) of regular expressions and relational expressions. Regular expressions are as in regexp(6). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions, using the operators = and !~. /re/ is a constant regular expression; any string (constant or variable) may be used as a regular expression, except in the position of an isolated regular expression in a pattern.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines from an occurrence of the first pattern though an occurrence of the second.
A relational expression is one of the following:

\[
\begin{align*}
expression & \text{ matchop} \ regular-expression \\
expression & \text{ relop} \ expression \\
expression & \text{ in} \ array-name \\
(expr, expr,...) & \text{ in} \ array-name
\end{align*}
\]

where a \text{ relop} is any of the six relational operators in C, and a \text{ matchop} is either \sim (matches) or \!\sim (does not match). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns \text{ BEGIN} and \text{ END} may be used to capture control before the first input line is read and after the last. \text{ BEGIN} and \text{ END} do not combine with other patterns.

Variable names with special meanings:

\begin{itemize}
\item \text{ CONVFMT} conversion format used when converting numbers (default \%.6g)
\item \text{ FS} regular expression used to separate fields; also settable by option \text{-Ffs}.
\item \text{ NF} number of fields in the current record
\item \text{ NR} ordinal number of the current record
\item \text{ FNR} ordinal number of the current record in the current file
\item \text{ FILENAME} the name of the current input file
\item \text{ RS} input record separator (default newline)
\item \text{ OFS} output field separator (default blank)
\item \text{ ORS} output record separator (default newline)
\item \text{ OFMT} output format for numbers (default \%.6g)
\item \text{ SUBSEP} separates multiple subscripts (default 034)
\item \text{ ARGC} argument count, assignable
\item \text{ ARGV} argument array, assignable; non-null members are taken as file names
\item \text{ ENVIRON} array of environment variables; subscripts are names.
\end{itemize}

Functions may be defined (at the position of a pattern-action statement) thus:

\[
\text{ function foo(a, b, c) \{ \ldots; \text{ return x } \} }
\]

Parameters are passed by value if scalar and by reference if array name; functions may be called recursively. Parameters are local to the function; all other variables are global. Thus local variables may be created by providing excess parameters in the function definition.

**EXAMPLES**

\[
\begin{align*}
\text{ length($0$) & > 72} \\
& \text{ Print lines longer than 72 characters.} \\
& \text{ \{ print $2$, $1$ \} } \\
& \text{ Print first two fields in opposite order.} \\
\text{ BEGIN \{} & \text{ FS = ",",[ \t]*[ \t]+ \} } \\
& \text{ \{} print $2$, $1$ \} \\
& \text{ Same, with input fields separated by comma and/or blanks and tabs.} \\
& \text{ \{} s += $1$ \} \\
\text{ END \{} & \text{ print "sum is", s, " average is", s/NR } \} \\
& \text{ Add up first column, print sum and average.} \\
\text{ /start/, /stop/} \\
& \text{ Print all lines between start/stop pairs.} \\
\text{ BEGIN \{} & \text{ # Simulate echo(1) } \\
& \text{ for (i = 1; i < ARGC; i++) printf "%s ", ARGV[i] } \\
& \text{ printf "\n" } \\
& \text{ exit \} }
\end{align*}
\]

**SOURCE**

/sys/src/cmd/awk

**SEE ALSO**

sed(1), regexp(6),

BUGS

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate " " to it.

The scope rules for variables in functions are a botch; the syntax is worse.

UTF is not always dealt with correctly, though awk does make an attempt to do so. The `split` function with an empty string as final argument now copes with UTF in the string being split.
NAME
basename - strip file name affixes

SYNOPSIS
basename [−d] string [suffix]

DESCRIPTION
Basename deletes any prefix ending in slash (/) and the suffix, if present in string, from string, and prints the result on the standard output.

The −d option instead prints the directory component, that is, string up to but not including the final slash. If the string contains no slash, a period and newline are printed.

SOURCE
/sys/src/cmd/basename.c
NAME
bc - arbitrary-precision arithmetic language

SYNOPSIS
bc [-cdls][ file ...]

DESCRIPTION
Bc is an interactive processor for a language that resembles C but provides arithmetic on numbers of arbitrary length with up to 100 digits right of the decimal point. It takes input from any files given, then reads the standard input.

The -d option enables debugging output. The -l option stands for the name of an arbitrary precision math library. The -s option suppresses the automatic display of calculation results; all output is via the print command.

The following syntax for bc programs is like that of C; L means letter a-z, E means expression, S means statement.

Lexical
comments are enclosed in /* */
newlines end statements

Names
simple variables: L
array elements: L[E]
The words ibase, obase, and scale

Other operands
arbitrarily long numbers with optional sign and decimal point.
(E)
sqrt(E)
length(E)
number of significant decimal digits
scale(E)
number of digits right of decimal point
L(E, ..., E)
function call

Operators
+  -   *  /  %  ^  (% is remainder; ^ is power)
++  --
==  <=  >=  !=  <  >
=  +=  -=  *= /= %= ^=

Statements
E
{ S; ... ; S }
print E
if ( E ) S
while ( E ) S
for ( E ; E ; E ) S
null statement
break
quit
"text"

Function definitions
define L( L, ... , L){
    auto L, ... , L
    S ; ... ; S
    return E
}

Functions in -l math library
s(x)  sine
c(x)  cosine
\[ e(x) \text{ exponential} \]
\[ l(x) \text{ log} \]
\[ a(x) \text{ arctangent} \]
\[ j(n, x) \text{ Bessel function} \]

All function arguments are passed by value.

The value of an expression at the top level is printed unless the main operator is an assignment or the \(-s\) command line argument is given. Text in quotes, which may include newlines, is always printed. Either semicolons or newlines may separate statements. Assignment to \texttt{scale} influences the number of digits to be retained on arithmetic operations in the manner of \texttt{dc}(1). Assignments to \texttt{ibase} or \texttt{obase} set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. Automatic variables are pushed down during function calls. In a declaration of an array as a function argument or automatic variable empty square brackets must follow the array name.

\textit{Bc} is actually a preprocessor for \texttt{dc}(1), which it invokes automatically, unless the \(-c\) (compile only) option is present. In this case the \texttt{dc} input is sent to the standard output instead.

**EXAMPLE**
Define a function to compute an approximate value of the exponential. Use it to print 10 values. (The exponential function in the library gives better answers.)

\[
\texttt{scale = 20} \\
\texttt{define e(x) \{} \\
\texttt{\quad auto a, b, c, i, s} \\
\texttt{\quad a = 1} \\
\texttt{\quad b = 1} \\
\texttt{\quad s = 1} \\
\texttt{\quad \textbf{for}(i=1; 1; i++) \{} \\
\texttt{\quad \quad a \*= x} \\
\texttt{\quad \quad b \*= i} \\
\texttt{\quad \quad c = a/b} \\
\texttt{\quad \quad \textbf{if}(c == 0) return s} \\
\texttt{\quad \quad s += c} \\
\texttt{\quad \}} \\
\texttt{\}} \\
\texttt{\textbf{for}(i=1; i<=10; i++) print e(i)} \\
\]

**FILES**

/sys/lib/bclib mathematical library

**SOURCE**
/sys/src/cmd/bc.y

**SEE ALSO**

\texttt{dc(1)}, \texttt{hoc(1)}

**BUGS**

No \&\&, \|\|, or ! operators.
A \texttt{for} statement must have all three \texttt{Es}.
A \texttt{quit} is interpreted when read, not when executed.
NAME
bind, mount, unmount - change name space

SYNOPSIS
bind [ option ... ] new old
mount [ option ... ] servename old [ spec ]
unmount [ new ] old

DESCRIPTION
Bind and mount modify the file name space of the current process and other processes in the same name space group (see fork(2)). For both calls, old is the name of an existing file or directory in the current name space where the modification is to be made.

For bind, new is the name of another (or possibly the same) existing file or directory in the current name space. After a successful bind, the file name old is an alias for the object originally named by new; if the modification doesn’t hide it, new will also still refer to its original file. The evaluation of new (see intro(2)) happens at the time of the bind, not when the binding is later used.

The servename argument to mount is the name of a file that, when opened, yields an existing connection to a file server. Almost always, servename will be a file in /srv (see srv(3)). In the discussion below, new refers to the file named by the new argument to bind or the root directory of the service available in servename after a mount. Either both old and new files must be directories, or both must not be directories.

Options control aspects of the modification to the name space:

(none) Replace the old file by the new one. Henceforth, an evaluation of old will be translated to the new file. If they are directories (for mount, this condition is true by definition), old becomes a union directory consisting of one directory (the new file).

-b Both files must be directories. Add the new directory to the beginning of the union directory represented by the old file.

-a Both files must be directories. Add the new directory to the end of the union directory represented by the old file.

-c This can be used in addition to any of the above to permit creation in a union directory. When a new file is created in a union directory, it is placed in the first element of the union that has been bound or mounted with the -c flag. If that directory does not have write permission, the create fails.

-C (Only in mount.) By default, file contents are always retrieved from the server. With this option, the kernel may instead use a local cache to satisfy read(5) requests for files accessible through this mount point. The currency of cached data for a file is verified at each open(5) of the file from this client machine.

-q Exit silently if the bind or mount operation fails.

Mount takes two additional options. The first, -k keypattern, constrains the set of factotum(4) keys used for an authenticated mount. The second, -n, causes mount to skip authentication entirely.

The spec argument to mount is passed in the attach(5) message to the server, and selects among different file trees served by the server.

The srv(3) service registry device, normally bound to /srv, is a convenient rendezvous point for services that can be mounted. After bootstrap, the file /srv/boot contains the communications port to the file system from which the system was loaded.

The effects of bind and mount can be undone with the unmount command. If two arguments are given to unmount, the effect is to undo a bind or mount with the same arguments. If only one argument is given, everything bound to or mounted upon old is unmounted.

EXAMPLES
To compile a program with the C library from July 16, 1992:
mount /srv/boot /n/dump dump
bind /n/dump/1992/0716/mips/lib/libc.a /mips/lib/libc.a
mk

SOURCE
/sys/src/cmd/bind.c
/sys/src/cmd/mount.c
/sys/src/cmd/unmount.c

SEE ALSO
bind(2), open(2), srv(3), srv(4)
NAME
bitsyload, light, pencal, keyboard, params, prompter - bitsy-specific utilities

SYNOPSIS
bitsy/bitsyload k|r [file]
bitsu/light [intensity]
bitsu/params [-f]
bitsu/pencal
bitsu/keyboard [-n]
bitsu/prompter [-n] file

DESCRIPTION
Bitsyload erases a section of flash memory on the Bitsy (iPAQ 3650 or 3830) and copies new information into it, using the format required by the Compaq boot loader. The required first argument is the destination, either k for /dev/flash/kernel or r for /dev/flash/ramdisk. The optional second argument is the name of the file to load. The default kernel file is /sys/src/9/bitsy/9bitsy and the default ramdisk file is /sys/src/9/bitsy/ramdisk.

Light sets the intensity of the display backlight. The values for intensity are:
on set intensity to maximum, the default
off turn off backlight
n sets the intensity to n, where n is a value between 0 and 128. Intensity 0 doesn't turn off the backlight, it just sets it to the dimmest value.

Pencal calibrates the display with the touch screen on a Bitsy. It loops prompting the user with crosses whose center that the user must touch with the stylus. After a consistent set of touches, it writes the calibration both to the kernel and to standard out. It is normally called by the bitsy's /bin/cpurc.

Params copies the contents of the file /dev/tmpparams, into the flash partition, /dev/flash/params, or if the -f flag it set copies in the opposite direction.

Keyboard creates a virtual on-screen keyboard and, unless the -n option is specified, a scribble area. A user inputs characters by tapping the keys or by drawing characters in the scribble area (see scribble(2)). It is usually run as the keyboard command for rio(1) using rio's -k option.

Prompter is a small editor used to configure parameters when a Bitsy boots. It displays the file and starts up a keyboard and scribble pad for input. Clicking with the stylus in the text selects where input characters will go. Pressing Button 5 (top left side of the Bitsy) or typing the Esc key on the keyboard causes prompter to write back the updated file and exit; Del causes prompter to exit without writing the file. The -n flag suppresses the scribble area.

EXAMPLE
Prompter, params, and calibrate are used in only one place, the Bitsy's /rc/bin/cpurc:

    # set variables
    ramfs
    bitsy/params -f
    if(! grep -s '^calibrate=' /tmp/tmpparams)
        bitsy/pencal >>/tmp/tmpparams
    if not {
        eval '{grep '^calibrate=' /tmp/tmpparams}
        echo calibrate $calibrate > '/m/mousectl'
    }
    bitsy/prompter /tmp/tmpparams
    bitsy/params

SOURCE
/sys/src/cmd/bitsy
NAME
  bundle - collect files for distribution

SYNOPSIS
  bundle file ...

DESCRIPTION
  Bundle writes on its standard output a shell script for rc(1) or a Bourne shell which, when executed, will recreate the original files. Its main use is for distributing small numbers of text files by mail(1).

  Although less refined than standard archives from ar(1) or tar(1), a bundle file is self-documenting and complete; little preparation is required on the receiving machine.

EXAMPLES
  bundle mkfile *.ch | mail kremvax!boris
  Send a makefile to Boris together with related .c and .h files. Upon receiving the mail, Boris may save the file sans postmark, say in gift/horse, then do
  cd gift; rc horse; mk

SOURCE
  /rc/bin/bundle

SEE ALSO
  ar(1), tar(1), mail(1)

BUGS
  Bundle will not create directories and is unsatisfactory for non-text files.
  Beware of gift horses.
NAME
cal - print calendar

SYNOPSIS
cal [ month ] [ year ]

DESCRIPTION
Cal prints a calendar. Month is either a number from 1 to 12, a lower case month name, or a lower case three-letter prefix of a month name. Year can be between 1 and 9999. If either month or year is omitted, the current month or year is used. If only one argument is given, and it is a number larger than 12, a calendar for all twelve months of the given year is produced; otherwise a calendar for just one month is printed. The calendar produced is that for England and her colonies.
Try
cal sep 1752

SOURCE
/sys/src/cmd/cal.c

BUGS
The year is always considered to start in January even though this is historically naive.
Beware that cal 90 refers to the early Christian era, not the 20th century.
NAME
   calendar - print upcoming events

SYNOPSIS
   calendar [−dy][−p days][ file ...]

DESCRIPTION
   Calendar reads the named files, default /usr/$user/lib/calendar, and writes to standard output any lines containing today's or tomorrow's date. Examples of recognized date formats are "4/11", "April 11", "Apr 11", "11 April", and "11 Apr". A special form may be used to represent weekly and monthly events: "Every Tuesday" "The third Wednesday" All comparisons are case insensitive.

If the −y flag is given, an attempt is made to match on year too. In this case, dates of the forms listed above will be accepted if they are followed by the current year (or last two digits thereof) or not a year — digits not followed by white space or non-digits.

If the −p flag is given, its argument is the number of days ahead to match dates. This flag is not repeatable, and it performs no special processing at the end of the week.

The −d flag enables debugging output.

On Friday and Saturday, events through Monday are printed.

To have your calendar mailed to you every day, use cron(8).

FILES
   /usr/$user/lib/calendar personal calendar

SOURCE
   /sys/src/cmd/calendar.c
NAME
cat, read - catenate files

SYNOPSIS
  cat [ file ...]
  read [-m] [-n nline] [ file ...]

DESCRIPTION
  *Cat* reads each *file* in sequence and writes it on the standard output. Thus
    cat file
  prints a file and
    cat file1 file2 >file3
  concatenates the first two files and places the result on the third.
If no *file* is given, *cat* reads from the standard input. Output is buffered in blocks matching the
input.
*Read* copies to standard output exactly one line from the named *file*, default standard input. It is
useful in interactive *rc(1)* scripts.
The *-m* flag causes it to continue reading and writing multiple lines until end of file; *-n* causes it
to read no more than *nline* lines.
*Read* always executes a single *write* for each line of input, which can be helpful when preparing
input to programs that expect line-at-a-time data. It never reads any more data from the input
than it prints to the output.

SOURCE
/sys/src/cmd/cat.c
/sys/src/cmd/read.c

SEE ALSO
  *cp(1)*

DIAGNOSTICS
  *Read* exits with status *EOF* on end of file or, in the *-n* case, if it doesn’t read *nlines* lines.

BUGS
  Beware of *cat a b >a* and *cat a b >b*, which destroy input files before reading them.
NAME
   cb - C program beautifier

SYNOPSIS
   cb [-js] [-l length][ file ... ]

DESCRIPTION
   Cb reads syntactically correct C programs from from its input or the given files, and writes them to
   its stdout with a more visually pleasing spacing and indentation. Cb understands no C++ syntax
   bar newline-terminated comments; and by default all user new-lines are preserved in the output.

   The options are:
   -j     Join split lines.
   -s     Print code in the so-called K&R style used in The C Programming Language.
   -l length
       Split lines that are longer than length.

SOURCE
   /sys/src/cmd/cb

BUGS
   Cb does not reformat structure initializers.
   Punctuation hidden in macros can cause indentation errors.
NAME
    chgrp - change file group

SYNOPSIS
    chgrp [-ou] group file ...

DESCRIPTION
    The group of each named file is changed to group, which should be a name known to the server holding the file.
    A file's group can be changed by the file's owner, if the owner is a member of the new group, or by the leader of both the file's current group and the new group.
    The -o and -u option are synonyms; they specify that the owner is to be set, rather than the group. They are ineffectual unless the file server is in the bootstrap state that permits changing file ownership.

SOURCE
    /sys/src/cmd/chgrp.c

SEE ALSO
    ls(1), chmod(1), stat(2)
NAME
chmod - change mode

SYNOPSIS
chmod mode file ...

DESCRIPTION
The mode of each named file is changed according to mode, which may be an octal number or a
symbolic change to the existing mode. A mode is an octal number constructed from the OR of the
following modes.
0400  read by owner
0200  write by owner
0100  execute (search in directory) by owner
0070  read, write, execute (search) by group
0077  read, write, execute (search) by others

A symbolic mode has the form:
[who] op permission

The who part is a combination of the letters u (for user’s permissions), g (group) and o (other). The
letter a stands for ugo. If who is omitted, the default is a.

Op can be + to add permission to the file’s mode, – to take away permission, and = to assign
permission absolutely (all other bits will be reset).

Permission is any combination of the letters r (read), w (write), x (execute), a (append only), l
(exclusive access), and t (temporary file).

Only the owner of a file or the group leader of its group may change the file’s mode.

SOURCE
/sys/src/cmd/chmod.c

SEE ALSO
ls(1), stat(2), stat(5)
NAME
cleanname - clean a path name

SYNOPSIS
cleanname [-d pwd] names ...

DESCRIPTION
For each file name argument, cleanname, by lexical processing only, prints the shortest equivalent string that names the same (possibly hypothetical) file. It eliminates multiple and trailing slashes, and it lexically interprets . and .. directory components in the name. If the -d option is present, unrooted names are prefixed with pwd/ before processing.

SOURCE
/sys/src/cmd/cleanname.c

SEE ALSO
cleanname(2).
NAME
  cmp - compare two files

SYNOPSIS
  cmp [ −lLs ] file1 file2 [ offset1 [ offset2 ]]

DESCRIPTION
  Cmp compares the two files and prints a message if the contents differ.
  The options are:
  −l  Print the byte number (decimal) and the differing bytes (hexadecimal) for each difference.
  −L  Print the line number of the first differing byte.
  −s  Print nothing for differing files, but set the exit status.
  If offsets are given, comparison starts at the designated byte position of the corresponding file. Offsets that begin with 0x are hexadecimal; with 0, octal; with anything else, decimal.

SOURCE
  /sys/src/cmd/cmp.c

SEE ALSO
  diff(1)

DIAGNOSTICS
  If a file is inaccessible or missing, the exit status is open. If the files are the same, the exit status is empty (true). If they are the same except that one is longer than the other, the exit status is EOF. Otherwise cmp reports the position of the first disagreeing byte and the exit status is differ.
NAME
col - column alignment

SYNOPSIS
col [-bfx]

DESCRIPTION
Col overlays lines to expunge reverse line feeds (ESC-7) and half line feeds (ESC-9 and ESC-8) as produced by nroff for .2C in ms(6) or man(6) and for tbl(1). Col is a pure filter. It normally emits only full line feeds; option -f (fine) allows half line feeds too. Option -b removes backspaces, printing just one of each pile of overstruck characters. Col normally converts white space to tabs; option -x overrides this feature. Other escaped characters and non-printing characters are ignored.

EXAMPLES
tbl file | nroff -ms | col | p
    Format some tables for printing on typewriters; use col to remove reverse line feeds, and paginate the output.

SEE ALSO
pr(1)

BUGS
Col can't back up more than 128 lines or handle more than 800 characters per line, and understands VT (013) as reverse line feed.
NAME
getmap, colors - display color map

SYNOPSIS
  colors [-rx]
  getmap [ colormap ]

DESCRIPTION
Colors presents a grid showing the colors in the current color map. If the display is true color, colors shows a grid of the RGBV color map (see color(6)).

Clicking mouse button 1 over a color in the grid will display the map index for that color, its red, green, and blue components, and the 32-bit hexadecimal color value as defined in allocimage(2).

If the -x option is specified, the components will also be listed in hexadecimal.

The -r option instead shows, in the same form, a grey-scale ramp.

A menu on mouse button 3 contains a single entry, to exit the program.

On 8-bit color-mapped displays, getmap loads the display's color map (default rgbv). The named colormap can be a file in the current directory or in the standard repository /lib/cmap. It can also be a string of the form gamma or gammaN, where N is a floating point value for the gamma, defining the contrast for a monochrome map. Similarly, rgamma and rgammaN define a reverse-video monochrome map. Finally, the names screen or display or vga are taken as synonyms for the current color map stored in the display hardware.

FILES
  /lib/cmap directory of color map files

SOURCE
  /sys/src/cmd/colors.c

SEE ALSO
  color(6)
NAME
   comm - select or reject lines common to two sorted files

SYNOPSIS
   comm [-123] file1 file2

DESCRIPTION
   Comm reads file1 and file2, which are in lexicographical order, and produces a three column output: lines only in file1; lines only in file2; and lines in both files. The file name – means the standard input.

   Flag 1, 2, or 3 suppresses printing of the corresponding column.

EXAMPLE
   comm -12 file1 file2
       Print lines common to two sorted files.

SOURCE
   /sys/src/cmd/comm.c

SEE ALSO
   sort(1), cmp(1), diff(1), uniq(1)
NAME
con, telnet, rx, hayes, xms, xmr - remote login, execution, and XMODEM file transfer

SYNOPSIS
con [-CdnrRsTv][-b baud][-l [user]][-S svc][-c cmd][net!]machine
telnet [-dCrn][-s svc][net!]machine
rx [-eTr][-l user][net!]machine [command–word ...]
hayes [-pv] number [device]
xms [-1p] file
xmr file

DESCRIPTION
Con connects to the computer whose network address is net!machine and logs in if possible. With no options, the account name used on the remote system is the same as that on the local system. Standard input and output go to the local machine.

Options are:
- **−b** sets the baud rate of a dial-up connection to *baud*.
- **−n** if the input is a file or pipe, do not hang up the connection when EOF is received, but instead wait for the remote end to hang up.
- **−l** with an argument causes *user* to be used as the account name on the remote system when performing BSD rlogin authentication. Without an argument this option disables automatic login and a normal login session ensues.
- **−C** forces cooked mode, that is, local echo.
- **−c** runs *cmd* as if it had been typed as a command from the escape mode.
- **−v** (verbose mode) causes information about connection attempts to be output to standard error. This can be useful when trying to debug network connectivity.
- **−d** causes debugging information to be output to standard error.
- **−r** suppresses printing of any carriage return followed by a new line. This is useful since carriage return is a printable character in Plan 9.
- **−R** translates newlines to carriage returns and *vice versa*.
- **−T** translates incoming carriage returns to newlines.
- **−s** strips received characters to 7 bits to forestall misinterpretation of ASCII with parity as UTF.
- **−S** Post a pipe as /szv/svc and connect it to standard input and output. This can be used with −n to create a standing connection that consolefs(4), for example, can then open. For telnet, this option is −s.

The control−\ character is a local escape. It prompts with >>>. Legitimate responses to the prompt are:
- **i** Send a quit [sic] signal to the remote machine.
- **q** Exit.
- **b** Send a break.
- **.** Return from the escape.
- **! cmd** Run the command with the network connection as its standard input and standard output. Standard error will go to the screen. This is useful for transmitting and receiving files over the connections using programs such as xms.
- **r** Toggle printing of carriage returns.

Telnet is similar to con, but uses the telnet protocol to communicate with the remote machine. It shares con’s −C, −d, −n, and −r options.

Rx executes one shell command on the remote machine as if logged in there, but with local standard input and output. A rudimentary shell environment is provided. If the target is a Plan 9 machine, $service there will be rx. Options are:
-e  a zero length message will not be written to the connection when standard input is closed.
-l  runs as user on the remote machine if the remote is a BSD machine.
-r  same as for con
-T  same as for con

Network addresses for both con and rx have the form network!machine. Supported networks are those listed in /net.

Hayes dials number on a Hayes-compatible modem, device. Under -p, it uses pulse dialing. Upon connecting, bytes are copied bidirectionally between the connection and standard input and output.

The commands xms and xmr respectively send and receive a single file using the XMODEM protocol. They use standard input and standard output for communication and are intended for use with con. The -1 option to xms causes it to use kilobyte packet size of 1024 bytes. The -p option causes it to print a progress message every ten kilobytes.

EXAMPLES

rx kremvax cat file1 >file2  
Copy remote file1 to local file2.

rx kremvax cat file1 '>'+file2'  
Copy remote file1 to remote file2.

eqn paper | rx kremvax troff −ms | rx deepthought lp  
Parallel processing: do each stage of a pipeline on a different machine.

SOURCE

/sys/src/cmd/rx.c
/sys/src/cmd/ip/telnet.c
/sys/src/cmd/con       for all other commands

SEE ALSO

cpu(1), ssh(1), telco(4)

BUGS

Con and telnet are merely obsolescent; the other commands are obsolete and deprecated.

Under rx, a program that should behave specially towards terminals may not: e.g., remote shells will not prompt. Also under rx, the remote standard error and standard output are combined and go inseparably to the local standard output. Rx will consume its standard input by copying it to the remote system, so redirect it from /dev/null if that's not what you want.
NAME
   cp, fcp, mv – copy, move files

SYNOPSIS
   cp [−gux ] file1 file2
   cp [−gux ] file ... directory
   fcp [−gux ] file1 file2
   fcp [−gux ] file ... directory
   mv file1 file2
   mv file ... directory

DESCRIPTION
   In the first form file1 is any name and file2 is any name except an existing directory. In the second
   form the commands copy or move one or more files into a directory under their original file
   names, as if by a sequence of commands in the first form. Thus cp f1 f2 dir is equivalent to
   cp f1 dir/f1; cp f2 dir/f2.

   Cp copies the contents of plain file1 to file2. The mode and owner of file2 are preserved if it
   already exists; the mode of file1 is used otherwise. The −x option sets the mode and modified
   time of file2 from file1; −g sets the group id; and −u sets the group id and user id (which is usu-
   ally only possible if the file server is in an administrative mode).

   Fcp behaves like cp but transfers multiple blocks in parallel while copying; it is noticeably faster
   than cp when the files involved are stored on servers connected over long–distance lines. It is only
   appropriate to use fcp with file servers that respect the offset in read(5) and write messages. This
   includes the disk–based file systems and ramfs but excludes most device file systems.

   Mv moves file1 to file2. If the files are in the same directory, file1 is just renamed; otherwise mv
   behaves like cp −x followed by rm file1. Mv will rename directories, but it refuses to move a direc-
   tory into another directory.

SOURCE
   /sys/src/cmd/cp.c
   /sys/src/cmd/fcp.c
   /sys/src/cmd/mv.c

SEE ALSO
   cat(1), dircp in tar(1), stat(2), read(5)

DIAGNOSTICS
   Cp, fcp, and mv refuse to copy or move files onto themselves.
NAME
cpp - C language preprocessor

SYNOPSIS
cpp [ option ... ] [ ifile [ ofile ]]

DESCRIPTION
Cpp interprets ANSI C preprocessor directives and does macro substitution. The input ifile and output ofile default to standard input and standard output respectively.

The options are:
-D name
-D name=def
-I dir Same as in 2c(1): add dir to the search for directives.
-M Generate no output except a list of include files in a form suitable for specifying dependencies to mk(1). Use twice to list files in angle brackets.
-N Turn off default include directories. All must be specified with −I, or in the environment variable include. Without this option, /$objtype/include and /sys/include are used as the last two searched directories for include directives, where $objtype is read from the environment.
-V Print extra debugging information.
-P Do not insert “#line” directives into the output.
++ Understand C++ comments.
−. Inhibit include search in the source’s directory.
−i Print the list of directories searched when #include is found. Last listed are searched first.

In the absence of the −P option, the processed text output is sprinkled with lines that show the original input line numbering:

#line linenumber "ifile"

The command reads the environment variable include and adds its (blank-separated) list of directories to the standard search path for directives. They are looked at before any directories specified with −I, which are looked at before the default directories.

The input language is as described in the ANSI C standard. The standard Plan 9 C compilers do not use cpp; they contain their own simple but adequate preprocessor, so cpp is usually superfluous.

FILES
/sys/include directory for machine-independent include files
/$objtype/include directory for machine-dependent include files

SOURCE
/sys/src/cmd/cpp

SEE ALSO
2c(1)
NAME
cpu - connection to CPU server

SYNOPSIS
cpu [-R | -O]

DESCRIPTION
Cpu starts an rc(1) running on the server machine, or the machine named in the $cpu environment variable if there is no –h option. Rc’s standard input, output, and error files will be /dev/cons in the name space where the cpu command was invoked. Normally, cpu is run in an rio(1) window on a terminal, so rc output goes to that window, and input comes from the keyboard when that window is current. Rc’s current directory is the working directory of the cpu command itself.

The name space for the new rc is an analogue of the name space where the cpu command was invoked: it is the same except for architecture-dependent bindings such as /bin and the use of fast paths to file servers, if available.

If a –u argument is present, cpu uses the argument as the remote user id.

If a –c argument is present, the remainder of the command line is executed by rc on the server, and then cpu exits.

If a –P argument is present, the patternfile is passed to exportfs(4) to control how much of the local name space will be exported to the remote system.

The –a command allows the user to specify the authentication mechanism used when connecting to the remote system. The two possibilities for auth-method are:
p9 This is the default. Authentication is done using the standard Plan 9 mechanisms, (see authsrv(6)). No user interaction is required.
netkey Authentication is done using challenge/response and a hand held authenticator or the netkey program (see passwd(1)). The user must encrypt the challenge and type the encryption back to cpu. This is used if the local host is in a different protection domain than the server or if the user wants to log into the server as a different user.

The –e option specifies an encryption and/or hash algorithm to use for the connection. If both are specified, they must be space separated and comprise a single argument, so they must be quoted if in a shell command. The default is rc4_256 encryption and sha1 hashing. See ssl(3) for details on possible algorithms. The argument clear specifies no encryption algorithm and can be used to talk to older versions of the cpu service.

The –k flag specifies a key pattern to use to restrict the keys selected by the auth_proxy call used for authentication.

The name space is built by running /usr/$user/lib/profile with the root of the invoking name space bound to /mnt/term. The service environment variable is set to cpu; the cmaptype and objtype environment variables reflect the server’s architecture.

The –R flag causes cpu to run the server (remote) side of the protocol. It is run from service files such as /bin/service/tcp17010. The –O flag is similar but simulates the pre–9P2000 version of the cpu protocol.

FILES
The name space of the terminal side of the cpu command is mounted, via exportfs(4), on the CPU side on directory /mnt/term. The files such as /dev/cons are bound to their standard locations from there.

SOURCE
/sys/src/cmd/cpu.c

SEE ALSO
rc(1), rio(1), exportfs(4)
BUGS

Binds and mounts done after the terminal `lib/profile` is run are not reflected in the new name space.

When using the `−a` option to ‘log in’ as another user, be aware that resources in the local name space will be made available to that user.
NAME

crop, iconv - frame, crop, and convert image

SYNOPSIS

crop [-b red green blue] [-c red green blue] [-i n | -x minx miny maxx maxy | -x dx | -y dy] [-t tx ty] [-b red green blue] [file]
iconv [-u] [-c chandesc] [file]

DESCRIPTION

Crop reads an image(6) file (default standard input), crops it, and writes it as a compressed image(6) file to standard output. There are two ways to specify a crop, by color value or by geometry. They may be combined in a single run of crop, in which case the color value crop will be done first.

The -c option takes a red-green-blue triplet as described in color(2). (For example, white is 255 255 255.) The corresponding color is used as a value to be cut from the outer edge of the picture; that is, the image is cropped to remove the maximal outside rectangular strip in which every pixel has the specified color.

The -i option insets the image rectangle by a constant amount, n, which may be negative to generate extra space around the image. The -x and -y options are similar, but apply only to the x or y coordinates of the image.

The -x option specifies an exact rectangle.

The -t option specifies that the image’s coordinate system should be translated by tx, ty as the last step of processing.

The -b option specifies a background color to be used to fill around the image if the cropped image is larger than the original, such as if the -i option is given a negative argument. This can be used to draw a monochrome frame around the image. The default color is black.

Iconv changes the format of pixels in the image file (default standard input) and writes the resulting image to standard output. Pixels in the image are converted according to the channel descriptor chandesc, (see image(6)). For example, to convert a 4-bit-per-pixel grey-scale image to an 8-bit-per-pixel color-mapped image, chandesc should be m8. If chandesc is not given, the format is unchanged. The output image is by default compressed; the -u option turns off the compression.

EXAMPLE

To crop white edges off the picture and add a ten-pixel pink border,
crop -c 255 255 255 -i -10 -b 255 150 150 imagefile > cropped

SOURCE

/sys/src/cmd/crop.c

SEE ALSO

image(6), color(2)

BUGS

Iconv should be able to do Floyd-Steinberg error diffusion or dithering when converting to small image depths.
NAME
date, clock - date and time

SYNOPSIS
date [ option ] [ seconds ]
clock

DESCRIPTION
Print the date, in the format
    Tue Aug 16 17:03:52 CDT 1977

The options are
−u Report Greenwich Mean Time (GMT) rather than local time.
−n Report the date as the number of seconds since the epoch, 00:00:00 GMT, January 1, 1970.

The conversion from Greenwich Mean Time to local time depends on the $timezone environment variable; see ctime(2).

If the optional argument seconds is present, it is used as the time to convert rather than the real time.

Clock draws a simple analog clock in its window.

FILES
/env/timezone Current timezone name and adjustments.
/adm/timezone A directory containing timezone tables.
/adm/timezone/local Default timezone file, copied by init(8) into /env/timezone.

SOURCE
/sys/src/cmd/date.c
/sys/src/cmd/clock.c
NAME
db - debugger

SYNOPSIS
db [ option ... ] [ textfile ] [ pid ]

DESCRIPTION
Db is a general purpose debugging program. It may be used to examine files and to provide a
controlled environment for the execution of Plan 9 programs.

A textfile is a file containing the text and initialized data of an executable program. A memfile is
the memory image of an executing process. It is usually accessed via the process id (pid) of the
process in /proc/pid/mem. A memfile contains the text, data, and saved registers and process
state. A map associated with each textfile or memfile supports accesses to instructions and data in
the file; see ‘Addresses’.

An argument consisting entirely of digits is assumed to be a process id; otherwise, it is the name
of a textfile. When a textfile is given, the textfile map is associated with it. If only a pid is given,
the textfile map is associated with /proc/pid/text. When a pid is given, the memfile map is
associated with /proc/pid/mem; otherwise it is undefined and accesses to the memfile are not
permitted.

Commands to db are read from the standard input and responses are to the standard output. The
options are
-k Use the kernel stack of process pid to debug the executing kernel process. If textfile is not
specified, file /Scputype/9type is used, where type is the second word in $terminal.
-w Create textfile and memfile if they don’t exist; open them for writing as well as reading.
-I path
Directory in which to look for relative path names in $< and $<< commands.

-machine
Assume instructions are for the given CPU type (any standard architecture name, such as
alpha or 386, plus mipsco and sunsparc, which cause disassembly to the
manufacturer’s syntax) instead of using the magic number to select the CPU type.

Most db commands have the following form:
[address] [, count] [command]

If address is present then the current position, called ‘dot’, is set to address. Initially dot is set to
0. Most commands are repeated count times with dot advancing between repetitions. The default
count is 1. Address and count are expressions. Multiple commands on one line must be separated
by ;

Expressions
Expressions are evaluated as long ints.
.
The value of dot.
+
The value of dot incremented by the current increment.
^The value of dot decremented by the current increment.
"The last address typed.

integer
A number, in decimal radix by default. The prefixes 0 and 0o and 0 (zero oh) force interpret-
ation in octal radix; the prefixes 0t and 0T force interpretation in decimal radix; the
prefixes 0x, OX, and # force interpretation in hexadecimal radix. Thus 020, 0o20, 0t16,
and #10 all represent sixteen.

integer . fraction
A single-precision floating point number.
’ c ’The 16–bit value of a character. \ may be used to escape a ‘.

<name
The value of name, which is a register name. The register names are those printed by the
symbol
A symbol is a sequence of upper or lower case letters, underscores or digits, not starting
with a digit. \ may be used to escape other characters. The location of the symbol is cal-
culated from the symbol table in textfile.

routine, name
The address of the variable name in the specified C routine. Both routine and name are
symbols. If name is omitted the value is the address of the most recently activated stack
frame corresponding to routine; if routine is omitted, the active procedure is assumed.

file: integer
The address of the instruction corresponding to the source statement at the indicated line
number of the file. If the source line contains no executable statement, the address of the
instruction associated with the nearest executable source line is returned. Files begin at
line 1. If multiple files of the same name are loaded, an expression of this form resolves to
the first file encountered in the symbol table.

(exp)
The value of the expression exp.

Monadic operators
*exp The contents of the location addressed by exp in memfile.
@exp The contents of the location addressed by exp in textfile.
exp Integer negation.
~exp Bitwise complement.
%exp When used as an address, exp is an offset into the segment named ublock; see
‘Addresses’.

Dyadic operators are left–associative and are less binding than monadic operators.

\(e1 + e2\) Integer addition.
\(e1 - e2\) Integer subtraction.
\(e1 * e2\) Integer multiplication.
\(e1 \% e2\) Integer division.
\(e1 \& e2\) Bitwise conjunction.
\(e1 | e2\) Bitwise disjunction.
\(e1 \# e2\) E1 rounded up to the next multiple of e2.

Commands
Most commands have the following syntax:

? f Locations starting at address in textfile are printed according to the format f.
/ f Locations starting at address in memfile are printed according to the format f.
= f The value of address itself is printed according to the format f.

A format consists of one or more characters that specify a style of printing. Each format character
may be preceded by a decimal integer that is a repeat count for the format character. If no format
is given then the last format is used.

Most format letters fetch some data, print it, and advance (a local copy of) dot by the number of
bytes fetched. The total number of bytes in a format becomes the current increment.

O Print two–byte integer in octal.
Q Print two–byte integer in signed octal.
D Print four–byte integer in decimal.
V Print eight-byte integer in decimal.
Z Print eight-byte integer in unsigned decimal.
X Print two-byte integer in hexadecimal.
Y Print eight-byte integer in hexadecimal.
U Print two-byte integer in unsigned decimal.
F Print as a single-precision floating point number.
B Print double-precision floating point.
C Print the addressed byte as a character. Printable ASCII characters are represented normally; others are printed in the form \xnn.
s Print the addressed characters, as a UTF string, until a zero byte is reached. Advance dot by the length of the string, including the zero terminator.
S Print a string using the escape convention (see C above).
r Print as UTF the addressed two-byte integer (run). Advance dot by the length of the string, including the zero terminator.
i Print as machine instructions. Dot is incremented by the size of the instruction.
I As i above, but print the machine instructions in an alternate form if possible: sunsparc and mipsco reproduce the manufacturers' syntax.
M Print the addressed machine instruction in a machine-dependent hexadecimal form.
a Print the value of dot in symbolic form. Dot is unaffected.
A Print the value of dot in hexadecimal. Dot is unaffected.
z Print the addressed value in symbolic form. Dot is advanced by the size of a machine address.
t When preceded by an integer, tabs to the next appropriate tab stop. For example, 8t moves to the next 8-space tab stop. Dot is unaffected.
n Print a newline. Dot is unaffected.
"..." Print the enclosed string. Dot is unaffected.
^ Dot is decremented by the current increment. Nothing is printed.
+ Dot is incremented by 1. Nothing is printed.
- Dot is decremented by 1. Nothing is printed.

Other commands include:

newline
Update dot by the current increment. Repeat the previous command with a count of 1.

[/\]L value mask
Words starting at dot are masked with mask and compared with value until a match is found. If L is used, the match is for a two-byte integer; L matches four bytes. If no match is found then dot is unchanged; otherwise dot is set to the matched location. If mask is omitted then ~0 is used.

[/\]W value ...
Write the two-byte value into the addressed location. If the command is W, write four bytes.

[/\]m s b e f...
New values for (b, e, f) in the segment named s are recorded. Valid segment names are text, data, or ublock. If less than three address expressions are given, the remaining parameters are left unchanged. If the list is terminated by ? or / then the file (textfile or memfile respectively) is used for subsequent requests. For example, /m? causes / to refer to textfile.

> name
Dot is assigned to the variable or register named.
! The rest of the line is passed to rc(1) for execution.
$ modifier

Miscellaneous commands. The available modifiers are:

< f Read commands from the file f. If this command is executed in a file, further commands in the file are not seen. If f is omitted, the current input stream is terminated. If a count is given, and is zero, the command is ignored.

<< f Similar to < except it can be used in a file of commands without causing the file to be closed. There is a (small) limit to the number of << files that can be open at once.

> f Append output to the file f, which is created if it does not exist. If f is omitted, output is returned to the terminal.

? Print process id, the condition which caused stopping or termination, the registers and the instruction addressed by pc. This is the default if modifier is omitted.

r Print the general registers and the instruction addressed by pc. Dot is set to pc.

R Like $r, but include miscellaneous processor control registers and floating point registers.

f Print floating-point register values as single-precision floating point numbers.

F Print floating-point register values as double-precision floating point numbers.

b Print all breakpoints and their associated counts and commands. ‘B’ produces the same results.

c Stack backtrace. If address is given, it specifies the address of a pair of 32-bit values containing the sp and pc of an active process. This allows selecting among various contexts of a multi-threaded process. If C is used, the names and (long) values of all parameters, automatic and static variables are printed for each active function. If count is given, only the first count frames are printed.

a Attach to the running process whose pid is contained in address.

e The names and values of all external variables are printed.

w Set the page width for output to address (default 80).

q Exit from db.

m Print the address maps.

k Simulate kernel memory management.

M machine
Set the machine type used for disassembling instructions.

: modifier

Manage a subprocess. Available modifiers are:

h Halt an asynchronously running process to allow breakpointing. Unnecessary for processes created under db, e.g. by : r.

b c Set breakpoint at address. The breakpoint is executed count-1 times before causing a stop. Also, if a command c is given it is executed at each breakpoint and if it sets dot to zero the breakpoint causes a stop.

d Delete breakpoint at address.

r Run textfile as a subprocess. If address is given the program is entered at that point; otherwise the standard entry point is used. Count specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command.

c s The subprocess is continued. If s is omitted or nonzero, the subprocess is sent the note that caused it to stop. If 0 is specified, no note is sent. (If the stop was due to a breakpoint or single-step, the corresponding note is elided before continuing.) Breakpoint skipping is the same as for r.

s s As for c except that the subprocess is single stepped for count machine instructions. If a note is pending, it is received before the first instruction is executed. If there is no current subprocess then textfile is run as a subprocess as for r. In this case no note can be sent; the remainder of the line is treated as arguments to the subprocess.

S s Identical to s except the subprocess is single stepped for count lines of C source. In optimized code, the correspondence between C source and the machine instructions is approximate at best.
The current subprocess, if any, is released by \textit{db} and allowed to continue executing
normally.

The current subprocess, if any, is terminated.

Display the pending notes for the process. If \textit{c} is specified, first delete \textit{c}’th pending
note.

**Addresses**

The location in a file or memory image associated with an address is calculated from a map
associated with the file. Each map contains one or more quadruples (\textit{t}, \textit{b}, \textit{e}, \textit{f}), defining a segment
named \textit{t} (usually, \textit{text}, \textit{data}, or \textit{ublock}) mapping addresses in the range \textit{b} through \textit{e} to the part
of the file beginning at offset \textit{f}. The memory model of a Plan 9 process assumes that segments are
disjoint. There can be more than one segment of a given type (e.g., a process may have more than
one text segment) but segments may not overlap. An address \textit{a} is translated to a file address by
finding a segment for which \textit{b} \leq \textit{a} < \textit{e}; the location in the file is then \textit{address} + \textit{f} - \textit{b}.

Usually, the text and initialized data of a program are mapped by segments called \textit{text} and \textit{data}.
Since a program file does not contain bss, stack or ublock data, these data are not mapped by the
data segment. The text segment is mapped similarly in a normal (i.e., non–kernel) \textit{memfile}. However,
the segment called \textit{data} maps memory from the beginning of the program’s data space to
the base of the ublock. This region contains the program’s static data, the bss, the heap and the
stack. A segment called \textit{ublock} maps the page containing its registers and process state.

Sometimes it is useful to define a map with a single segment mapping the region from 0 to
0xFFFFFFFF; a map of this type allows the entire file to be examined without address translation.

Registers are saved at a machine–dependent offset in the ublock. It is usually not necessary to
know this offset; the \texttt{$r, \$R, \$f, and \$F} commands calculate it and display the register contents.

The \texttt{$m} command dumps the currently active maps. The \texttt{?m and /m commands modify the seg-
ment parameters in the \textit{textfile} and \textit{memfile} maps, respectively.

**EXAMPLES**

To set a breakpoint at the beginning of \texttt{write()} in extant process 27:

% \texttt{db 27}
% :h
% \texttt{write:b}
% :c

To examine the Plan 9 kernel stack for process 27:

% \texttt{db –k 27}
% $C

Similar, but using a kernel named \texttt{test}:

% \texttt{db –k test 27}
% $C

To set a breakpoint at the entry of function \texttt{parse} when the local variable \texttt{argc} in \texttt{main} is equal
to 1:

\texttt{parse:b *main.argv–1=X}

This prints the value of \texttt{argv–1} which as a side effect sets dot; when \texttt{argc} is one the breakpoint
will fire. Beware that local variables may be stored in registers; see the BUGS section.

Debug process 127 on remote machine \texttt{kremvax}:

% \texttt{import kremvax /proc}
% \texttt{db 127}
% $C

**FILES**

/\texttt{proc/*/text}
/\texttt{proc/*/mem}
/\texttt{proc/*/ctl}
/\texttt{proc/*/note}
SEE ALSO
acid(1), nm(1), proc(3)

SOURCE
/sys/src/cmd/db

DIAGNOSTICS
Exit status is null, unless the last command failed or returned non-null status.

BUGS
Examining a local variable with routine.name returns the contents of the memory allocated for the variable, but with optimization (suppressed by the –N compiler flag) variables often reside in registers. Also, on some architectures, the first argument is always passed in a register.

Variables and parameters that have been optimized away do not appear in the symbol table, returning the error bad local variable when accessed by db.

Because of alignment incompatibilities, Motorola 68000 series machines can not be debugged remotely from a processor of a different type.

Breakpoints should not be set on instructions scheduled in delay slots. When a program stops on such a breakpoint, it is usually impossible to continue its execution.
NAME
dc - desk calculator

SYNOPSIS
dc [ file]

DESCRIPTION
Dc is an arbitrary precision desk calculator. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of dc is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

text
The value of the number is pushed on the stack. A number is an unbroken string of the digits 0–9A–F or 0–9a–f. A hexadecimal number beginning with a lower case letter must be preceded by a zero to distinguish it from the command associated with the letter. It may be preceded by an underscore _ to input a negative number. Numbers may contain decimal points.

s
Pop the top of the stack and store into a register named x, where x may be any character. Under operation S register x is treated as a stack and the value is pushed on it.

1
Push the value in register x onto the stack. The register x is not altered. All registers start with zero value. Under operation L register x is treated as a stack and its top value is popped onto the main stack.

d
Duplicate the top value on the stack.

p
Print the top value on the stack. The top value remains unchanged. P interprets the top of the stack as an text string, removes it, and prints it.

f
Print the values on the stack.

q
Q
Exit the program. If executing a string, the recursion level is popped by two. Under operation Q the top value on the stack is popped and the string execution level is popped by that value.

x
Treat the top element of the stack as a character string and execute it as a string of dc commands.

X
Replace the number on the top of the stack with its scale factor.

[ ... ]
Put the bracketed text string on the top of the stack.

<x
Pop and compare the top two elements of the stack. Register x is executed if they obey the stated relation.

>?
x
Replace the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

!
Interpret the rest of the line as a shell command.

c
Clear the stack.

i
The top value on the stack is popped and used as the number base for further input.
I Push the input base on the top of the stack.
O The top value on the stack is popped and used as the number base for further output. In bases larger than 10, each ‘digit’ prints as a group of decimal digits.
O Push the output base on the top of the stack.
K Pop the top of the stack, and use that value as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
Z Push the stack level onto the stack.
Z Replace the number on the top of the stack with its length.
? A line of input is taken from the input source (usually the terminal) and executed.
; : Used by bc for array operations.
The scale factor set by K determines how many digits are kept to the right of the decimal point. If s is the current scale factor, sa is the scale of the first operand, sb is the scale of the second, and b is the (integer) second operand, results are truncated to the following scales.

\[+,- \max(sa, sb)\]
\[\times \min(sa+sb, \max(sa, sb))\]
\[/ \begin{cases} s \text{ so that divisor} \times \text{quotient} + \text{remainder}; \text{remainder has sign of dividend} \\ \min(sa \times \lvert b \rvert, \max(s, sa)) \end{cases}\]
\[^{\text{v}} \max(s, sa)\]

EXAMPLES
Print the first ten values of n!
\[
\begin{align*}
[lal+dsa^plal0>y] & sy \\
0 & sal \\
& lyx
\end{align*}
\]

SOURCE
/sys/src/cmd/dc.c

SEE ALSO
bc(1), hoc(1)

DIAGNOSTICS
x is unimplemented, where x is an octal number: an internal error.
‘Out of headers’ for too many numbers being kept around.
‘Nesting depth’ for too many levels of nested execution.

BUGS
When the input base exceeds 16, there is no notation for digits greater than $F$.
Past its time.
NAME
   dd - convert and copy a file

SYNOPSIS
   dd [ option value ] ...

DESCRIPTION
   DD copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O. The options are

   -if f   Open file f for input.
   -of f   Open file f for output.
   -ibs n  Set input block size to n bytes (default 512).
   -obs n  Set output block size (default 512).
   -bs n   Set both input and output block size, superseding ibs and obs. If no conversion is specified, preserve the input block size instead of packing short blocks into the output buffer. This is particularly efficient since no in-core copy need be done.
   -cbs n  Set conversion buffer size.
   -skip n Skip n input records before copying.
   -isect n Seek n records forward on input file before copying.
   -files n Catenate n input files (useful only for magnetic tape or similar input device).
   -oseek n Seek n records from beginning of output file before copying.
   -count n Copy only n input records.
   -trunc n By default, dd truncates the output file when it opens it; -trunc 0 opens it without truncation.
   -quiet n By default, dd prints the number of blocks read and written once it is finished.
               -quiet 1 silences this summary.
   -conv ascii Convert EBCDIC to ASCII.
   ecbic     Convert ASCII to EBCDIC.
   ibm      Like ecbic but with a slightly different character map.
   block    Convert variable length ASCII records to fixed length.
   unblock  Convert fixed length ASCII records to variable length.
   lcase Map alphabatics to lower case.
   ucase Map alphabatics to upper case.
   swab    Swap every pair of bytes.
   noerror Do not stop processing on an error.
   sync Pad every input record to ibs bytes.

Where sizes are specified, a number of bytes is expected. A number may end with k or b to specify multiplication by 1024 or 512 respectively; a pair of numbers may be separated by x to indicate a product. Multiple conversions may be specified in the style: -conv ecbic,ucase.

Cbs is used only if ascii, unblock, ecbic, ibm, or block conversion is specified. In the first two cases, n characters are copied into the conversion buffer, any specified character mapping is done, trailing blanks are trimmed and new-line is added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer and blanks are added to make up an output record of size n. If cbs is unspecified or zero, the ascii, ecbic, and ibm options convert the character set without changing the block structure of the input file; the unblock and block options become a simple file copy.
SOURCE
   /sys/src/cmd/dd.c
SEE ALSO
   cp(1)
DIAGNOSTICS
   Dd reports the number of full + partial input and output blocks handled.
NAME
delkey - delete keys from factotum

SYNOPSIS
delkey [ -f ] pattern

DESCRIPTION
Delkey shows the user each key stored in factotum(4) and matching the pattern, prompting for
whether the key should be deleted. At each prompt, typing a response beginning with y deletes
the key, typing a response beginning with q aborts the listing, and any other response skips over
the key.
The -f option disables the prompting; all keys matching the pattern are deleted.
When run on a CPU server, delkey uses the terminal’s factotum, if present, instead of the server’s
factotum.

FILES
/mnt/term/mnt/factotum
    First choice for factotum to use
/mnt/factotum
    Second choice

SOURCE
/rc/bin/delkey
NAME
deroff, delatex - remove formatting requests

SYNOPSIS
deroff [ option ... ] file ...
delatex file

DESCRIPTION
Deroff reads each file in sequence and removes all nroff and troff requests and non-text arguments, backslash constructions, and constructs of preprocessors such as eqn(1), pic(1), and tbl(1). Remaining text is written on the standard output. Deroff follows files included by .so and .nx commands; if a file has already been included, a .so for that file is ignored and a .nx terminates execution. If no input file is given, deroff reads from standard input.

The options are
-w  Output a word, one 'word' (string of letters, digits, and properly embedded ampersands and apostrophes, beginning with a letter) per line. Other characters are skipped. Otherwise, the output follows the original, with the deletions mentioned above.
--  Like -w, but consider underscores to be alphanumeric rather than punctuation.
-i  Ignore .so and .nx requests.
-ms Remove titles, attachments, etc., as well as ordinary troff constructs, from ms(6) or mm documents.
-mm Same as -mm, but remove lists as well.
Delatex does for tex and latex (see tex(1)) files what deroff -wi does for troff files.

SOURCE
/sys/src/cmd/deroff.c
/sys/src/cmd/tex/local/delatex.c

SEE ALSO
troff(1), tex(1), spell(1)

BUGS
These filters are not complete interpreters of troff or tex. For example, macro definitions containing \$ cause chaos in deroff when the popular $$ delimiters for eqn are in effect.
Text inside macros is emitted at place of definition, not place of call.
NAME
diff – differential file comparator

SYNOPSIS
diff [-abcefmnrw] file1 ... file2

DESCRIPTION
Diff tells what lines must be changed in two files to bring them into agreement. If one file is a
directory, then a file in that directory with basename the same as that of the other file is used. If
both files are directories, similarly named files in the two directories are compared by the method
of diff for text files and cmp(1) otherwise. If more than two file names are given, then each argument
is compared to the last argument as above. The -r option causes diff to process similarly
named subdirectories recursively. When processing more than one file, diff prefixes file differences
with a single line listing the two differing files, in the form of a diff command line. The -m flag causes this behavior even when processing single files.

The normal output contains lines of these forms:

n1 a n3,n4
n1,n2 c n3
n1,n2 d n3

These lines resemble ed commands to convert file1 into file2. The numbers after the letters pertain
to file2. In fact, by exchanging ‘a’ for ‘d’ and reading backward one may ascertain equally how to
convert file2 into file1. As in ed, identical pairs where n1 = n2 or n3 = n4 are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by ‘<’, then
all the lines that are affected in the second file flagged by ‘>’.

The -b option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to
compare equal. The -w option causes all white-space to be removed from input lines before applying the difference algorithm.

The -n option prefixes each range with file: and inserts a space around the a, c, and d verbs. The -e option produces a script of a, c and d commands for the editor ed, which will recreate file2 from file1. The -E option produces a similar script, not useful with ed, in the opposite order. It may, however, be useful as input to a stream-oriented post-processor.

The -c option includes three lines of context around each change, merging changes whose contexts overlap. In this mode, diff prints – and + instead of < and > because the former are easier to distinguish when mixed. The -a flag displays the entire file as context.

Except in rare circumstances, diff finds a smallest sufficient set of file differences.

FILES
/tmp/diff[12]

SOURCE
/sys/src/cmd/diff

SEE ALSO
cmp(1), comm(1), ed(1), idiff(1)

DIAGNOSTICS
Exit status is the empty string for no differences, some for some, and error for trouble.

BUGS
Editing scripts produced under the -e or -E option are naive about creating lines consisting of a
single ‘.’.

When running diff on directories, the notion of what is a text file is open to debate.
NAME
doc2txt, doc2ps, wdoc2txt, xls2txt, olefs, mswordstrings, msexceltables - extract printable text from Microsoft documents

SYNOPSIS
doc2txt [ file.doc ]
doc2ps [ file.doc ]
wdoc2txt [ file.doc ]
xls2txt [ file.xls ]
aux/olefs [-m mtpt] file.doc
aux/mswordstrings mtpt/WordDocument
aux/msexceltables [-qaDnt] [-d delim] [-c column-range] [-w worksheet-range]
mtpt/Workbook

DESCRIPTION
Doc2txt is an rc(1) script that uses olefs and mswordstrings to extract the printable text from the body of a Microsoft Word document and write it on the standard output. Doc2ps is similar, but emits PostScript corresponding to the document. Wdoc2txt is similar to doc2txt, but uses plumbl(1) to send the output to a new acme(1) window instead. Xls2txt performs a similar function for Microsoft Excel documents.

Microsoft Office documents are stored in OLE (Object Linking and Embedding) format, which is a scaled down version of Microsoft’s FAT file system. Olefs presents the contents of an MS Office document as a file system on mtpt, which defaults to /mnt/doc. Mswordstrings or msexceltables may then be used to parse the files inside, extracting a text stream. Msexceltables may be given options to control the formatting of its output.

- a Attempt conversion of non-tabular sheets in the workbook (charts).
- d delim Sets the inter-field delimiter to the string delim, by default a single space.
- D Enables debugging output.
- c range Range is a comma-separated list of column numbers and ranges. Ranges are separated by dashes. Limit processing to just those columns named; by default all columns are output.
- n Disables field padding to column width.
- q Disable quoting of textural fields (see quote(2)).
- t Truncate fields to the column width.
- w range Range is a comma-separated list of worksheet numbers and ranges, this limits the sheets output using the same syntax as the -c option above. Suppressed chart pages are always included in the sheet count.

EXAMPLE
Extract pieces of an MS Excel spreadsheet.
aux/olefs report.xls
msexceltables -q -w 1,7,9–14 -c 3–5 -n -d '@' /mnt/doc/Workbook > rpt.txt
unmount /mnt/doc

SOURCE
/rc/bin doc2txt, doc2ps, wdoc2txt, and xls2txt
/sys/src/cmd/aux the others

SEE ALSO
strings(1)
“Microsoft Word 97 Binary File Format”, at Microsoft’s developer (MSDN) home page.
“LAOLA Binary Structures”, http://user.cs.tu-berlin.de/~schwartz/pmh
“OpenOffice.Org’s Excel Documentation”,
http://sc.openoffice.org/excelfileformat.pdf
NAME
dotype – intuit command line for formatting a document

SYNOPSIS
doctype [−n][−T dev][file]...

DESCRIPTION
Doctype examines a troff(1) input file to deduce the appropriate text formatting command and
prints it on standard output. Doctype recognizes input for troff(1), related preprocessors like
eqn(1), and the ms(6) and mm macro packages.

Option −n invokes nroff instead of troff. The −T option is passed to troff.

EXAMPLES
eval '{doctype chapter.?} | lp
 Typeset files named chapter.0, chapter.1, ...

SOURCE
/rc/bin/doctype

SEE ALSO
troff(1), eqn(1), tbl(1), pic(1), grap(1), ms(6), man(6)

BUGS
In true A.I. style, its best guesses are inspired rather than accurate.
NAME
  du - disk usage
SYNOPSIS
  du [−aefhnqstu] [−b size] [−p SI−prefix] [file ...]
DESCRIPTION
  Du gives the number of Kbytes allocated to data blocks of named files and, recursively, of files in named directories. It assumes storage is quantized in units of 1024 bytes (Kbytes) by default. Other values can be set by the −b option; size is the number of bytes, optionally suffixed k to specify multiplication by 1024. If file is missing, the current directory is used. The count for a directory includes the counts of the contained files and directories.
  The −a option prints the number of blocks for every file in a directory. Normally counts are printed only for contained directories.
  The −f option suppresses the printing of warning messages.
  The −n option prints the size in bytes and the name of each file; it sets −a.
  The −t option prints, in the format of du −n, the modified time of each file rather than the size.
  If the options −tu are specified then the accessed time is printed.
  The −q option prints, in the format of du −n, the QID path of each file rather than the size.
  The −s option causes du to descend the hierarchy as always, but to print only a summary line for each file.
  The −e option causes du to print values (sizes, times or QID paths) in ‘scientific notation’ via print(2)’s %g.
  The −h option causes du to print values (sizes, times or QID paths) in scientific notation, scaled to less than 1024, and with a suitable SI prefix (e.g., G for binary gigabytes).
  The −p option causes du to print values (sizes, times or QID paths) in units of SI−prefix. Case is ignored when looking up SI−prefix. An empty SI−prefix corresponds to a scale factor of 1 (e.g., print sizes in bytes).
EXAMPLES
  Print the size of /tmp in fractional binary gigabytes:
    % du −sepg /tmp
    .6960154 /tmp
  Print the size of /tmp in bytes and in scientific notation:
    % du −sep ’ ’ /tmp
    7.473408e+08 /tmp
SOURCE
  /sys/src/cmd/du.c
NAME
  echo - print arguments

SYNOPSIS
  echo [-n] [arg ...]

DESCRIPTION
  Echo writes its arguments separated by blanks and terminated by a newline on the standard output. Option \(-n\) suppresses the newline.

SOURCE
  /sys/src/cmd/echo.c

DIAGNOSTICS
  If echo draws an error while writing to standard output, the exit status is write error. Otherwise the exit status is empty.
NAME
  ecp – fast copy, handling errors

SYNOPSIS
  ecp [ -bcpxZ ] [ -B block-size ] [ -e max-errors ] [ -i issect ] [ -o ossect ] [ -s sector-size ]
    sectors input output

DESCRIPTION
  Ecp copies sectors disk sectors of the specified input file to the specified output file. Ecp copies
  multiple sectors (a ‘block’) at a time for speed. When ecp encounters an I/O error, it transfers the
current block again, assuming the file is seekable, one sector at a time, prints the sector number(s)
of the error(s), and continues copying.

Options are:
  -b reblock input on short reads; this was used mainly when reading a pipe on standard input on
    4.2+BSD systems.
  -B sets the block size (16,384 bytes by default) to block-size.
  -c ask for confirmation on /dev/console before starting the copy.
  -e sets a maximum number of consecutive I/O errors to permit at the beginning of the copy
    before quitting to max-errors. Lots of consecutive errors may indicate a deeper problem, such
    as missing media. By default there is no limit.
  -i seeks to sector issect (assuming zero-origin) before beginning input.
  -o seeks to sector ossect (assuming zero-origin) before beginning output.
  -p print reassuring progress reports; helpful mainly when dealing with cranky hardware.
  -r copy sector groups in reverse order, assuming the files are seekable; this is most useful when
    input and output overlap.
  -s sets the sector size (512 bytes by default) to sector-size.
  -v verify the copy by rereading the input and output files after copying all sectors. This is
    intended to force the disk to deliver the actual data written on it rather than some cached
    copy. The locations of any differences are printed.
  -Z ‘Swizzle’ the input: stir the bits around in some fashion. Intended for diagnosing bad disks by
    copying a disk to itself a few times with swizzling on (to defeat caching in operating systems
    or disk controllers).

SEE ALSO
  fc in cp(1), dd(1), dup(3)

BUGS
  -i, -o, -r, -v and error retries only work on devices capable of seeking.
  The set of options reflects decades of experience dealing with troublesome hardware.
  If the input file is a tape and the last record on the tape before a file mark is less than blocksize
  bytes long, then ecp will read through past the file mark and into the next file.
NAME
ed - text editor
SYNOPSIS
   ed [-] [-o] [ file ]
DESCRIPTION
   Ed is a venerable text editor.

   If a file argument is given, ed simulates an e command (see below) on that file: it is read into ed’s buffer so that it can be edited. The options are

   - Suppress the printing of character counts by e, I, and w commands and of the confirming ! by ! commands.
   -o (for output piping) Write all output to the standard error file except writing by w commands.  If no file is given, make /Ed/1 the remembered file; see the e command below.

   Ed operates on a ‘buffer’, a copy of the file it is editing; changes made in the buffer have no effect on the file until a w (write) command is given.  The copy of the text being edited resides in a temporary file called the buffer.

   Commands to ed have a simple and regular structure: zero, one, or two addresses followed by a single character command, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer.  Missing addresses are supplied by default.

   In general, only one command may appear on a line.  Certain commands allow the addition of text to the buffer.  While ed is accepting text, it is said to be in input mode.  In this mode, no commands are recognized; all input is merely collected.  Input mode is left by typing a period . alone at the beginning of a line.

   Ed supports the regular expression notation described in regexp(6).  Regular expressions are used in addresses to specify lines and in one command (see s below) to specify a portion of a line which is to be replaced.  If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by ‘\’.  This also applies to the character bounding the regular expression (often /) and to \ itself.

   To understand addressing in ed it is necessary to know that at any time there is a current line.  Generally, the current line is the last line affected by a command; however, the exact effect on the current line is discussed under the description of each command.  Addresses are constructed as follows.

   1.  The character ., customarily called ‘dot’, addresses the current line.
   2.  The character $ addresses the last line of the buffer.
   3.  A decimal number n addresses the n–th line of the buffer.
   4.  ’x addresses the line marked with the name x, which must be a lower–case letter.  Lines are marked with the k command.
   5.  A regular expression enclosed in slashes (/) addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression.  If necessary the search wraps around to the beginning of the buffer.
   6.  A regular expression enclosed in queries ? addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression.  If necessary the search wraps around to the end of the buffer.
   7.  An address followed by a plus sign + or a minus sign – followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines.  The plus sign may be omitted.
   8.  An address followed by + (or –) followed by a regular expression enclosed in slashes specifies the first matching line following (or preceding) that address.  The search wraps around if necessary.  The + may be omitted, so 0/x/ addresses the first line in the buffer with an x.  Enclosing the regular expression in ? reverses the search direction.
9. If an address begins with + or − the addition or subtraction is taken with respect to the current line; e.g. −5 is understood to mean −5.
10. If an address ends with + or −, then 1 is added (resp. subtracted). As a consequence of this rule and rule 9, the address − refers to the line before the current line. Moreover, trailing + and − characters have cumulative effect, so −− refers to the current line less 2.
11. To maintain compatibility with earlier versions of the editor, the character ^ in addresses is equivalent to −.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma , . They may also be separated by a semicolon ; . In this case the current line is set to the previous address before the next address is interpreted. If no address precedes a comma or semicolon, line 1 is assumed; if no address follows, the last line of the buffer is assumed. The second address of any two-address sequence must correspond to a line following the line corresponding to the first address.

In the following list of ed commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default. ‘Dot’ means the current line.

( . ) a
<text>
  . Read the given text and append it after the addressed line. Dot is left on the last line input, if there were any, otherwise at the addressed line. Address 0 is legal for this command; text is placed at the beginning of the buffer.

( . . ) b[+-][pagesize][p1n]
  Browse. Print a ‘page’, normally 20 lines. The optional + (default) or − specifies whether the next or previous page is to be printed. The optional pagesize is the number of lines in a page. The optional p, n, or 1 causes printing in the specified format, initially p. Pagesize and format are remembered between b commands. Dot is left at the last line displayed.

( . . ) c
<text>
  . Change. Delete the addressed lines, then accept input text to replace these lines. Dot is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

( . . ) d
  Delete the addressed lines from the buffer. Dot is set to the line following the last line deleted, or to the last line of the buffer if the deleted lines had no successor.

e filename
  Edit. Delete the entire contents of the buffer; then read the named file into the buffer. Dot is set to the last line of the buffer. The number of characters read is typed. The file name is remembered for possible use in later e, i, or w commands. If filename is missing, the remembered name is used.

E filename
  Unconditional e; see ‘q’ below.

f filename
  Print the currently remembered file name. If filename is given, the currently remembered file name is first changed to filename.

(1, $) g/regular expression/command list
(1, $) g/regular expression/
(1, $) g/regular expression
  Global. First mark every line which matches the given regular expression. Then for every such line, execute the command list with dot initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must end with \ . The ‘.’ terminating input
mode for an a, i, c command may be omitted if it would be on the last line of the com-
mand list. The commands g and v are not permitted in the command list. Any character
other than space or newline may be used instead of / to delimit the regular expression.
The second and third forms mean \g/regular expression/p.

(. )i
  Insert the given text before the addressed line. Dot is left at the last line input, or, if there
were none, at the line before the addressed line. This command differs from the a com-
mand only in the placement of the text.

( ,.+1)j
  Join the addressed lines into a single line; intermediate newlines are deleted. Dot is left at
the resulting line.

( . )kx
  Mark the addressed line with name x, which must be a lower–case letter. The address form
  ' x then addresses this line.

( . )l
  List. Print the addressed lines in an unambiguous way: a tab is printed as \t, a backspace
as \b, backslashes as \", and non–printing characters as a backslash, an x, and four hex-
decimal digits. Long lines are folded, with the second and subsequent sub–lines indented
one tab stop. If the last character in the line is a blank, it is followed by \n. An 1 may be
appended, like p, to any non–I/O command.

( . )ma
  Move. Reposition the addressed lines after the line addressed by a. Dot is left at the last
moved line.

( . )n
  Number. Perform p, prefixing each line with its line number and a tab. An n may be
appended, like p, to any non–I/O command.

( . )p
  Print the addressed lines. Dot is left at the last line printed. A p appended to any non–I/O
command causes the then current line to be printed after the command is executed.

( . )P
  This command is a synonym for p.

q
  Quit the editor. No automatic write of a file is done. A q or e command is considered to
be in error if the buffer has been modified since the last w, q, or e command.

Q
  Quit unconditionally.

($)r filename
  Read in the given file after the addressed line. If no filename is given, the remembered file
name is used. The file name is remembered if there were no remembered file name
already. If the read is successful, the number of characters read is printed. Dot is left at
the last line read from the file.

( . . )sn/regular expression/replacement/

( . . )sn/regular expression/replacement/g

( . . )sn/regular expression/replacement
  Substitute. Search each addressed line for an occurrence of the specified regular expres-
sion. On each line in which n matches are found ( n defaults to 1 if missing), the nth
matched string is replaced by the replacement specified. If the global replacement indica-
tor g appears after the command, all subsequent matches on the line are also replaced.
It is an error for the substitution to fail on all addressed lines. Any character other than
space or newline may be used instead of / to delimit the regular expression and the
replacement. Dot is left at the last line substituted. The third form means
sn/regular expression/replacement/p. The second / may be omitted if the replacement
is empty.

An ampersand & appearing in the replacement is replaced by the string matching the regu-
lar expression. The characters \n, where n is a digit, are replaced by the text matched by
the n–th regular subexpression enclosed between ( and ). When nested parenthesized
subexpressions are present, \( n \) is determined by counting occurrences of ( starting from
the left.

A literal &, /, \ or newline may be included in a replacement by prefixing it with \

\( (. . .) t a \)
Transfer. Copy the addressed lines after the line addressed by \( a \). Dot is left at the last line
of the copy.

\( (. . .) u \)
Undo. Restore the preceding contents of the first addressed line (sic), which must be the
last line in which a substitution was made (double sic).

\( (1, \$) v / \text{regular expression} / \text{command list} \)
This command is the same as the global command g except that the command list is ex-
cuted with dot initially set to every line except those matching the regular expression.

\( (1, \$) w \text{filename} \)
Write the addressed lines to the given file. If the file does not exist, it is created with mode
666 (readable and writable by everyone). If no \text{filename} is given, the remembered file
name, if any, is used. The file name is remembered if there were no remembered file name
already. Dot is unchanged. If the write is successful, the number of characters written is
printed.

\( (1, \$) W \text{filename} \)
Perform w, but append to, instead of overwriting, any existing file contents.

\( ($) = \)
Print the line number of the addressed line. Dot is unchanged.

! \text{shell command}
Send the remainder of the line after the ! to rc(1) to be interpreted as a command. Dot is
unchanged.

\( (. +1) < \text{newline} > \)
An address without a command is taken as a p command. A terminal / may be omitted
from the address. A blank line alone is equivalent to .+1p; it is useful for stepping
through text.

If an interrupt signal (DEL) is sent, \text{ed} prints a ? and returns to its command level.
When reading a file, \text{ed} discards NUL characters and all characters after the last newline.

\textbf{FILES}
/tmp/e*
ed.hup work is saved here if terminal hangs up

\textbf{SOURCE}
/sys/src/cmd/ed.c

\textbf{SEE ALSO}
sam(1), sed(1), regexp(6)

\textbf{DIAGNOSTICS}
? name for inaccessible file; ?TMP for temporary file overflow; ? for errors in commands or other
overflows.
NAME
  emacs - editor macros

SYNOPSIS
  emacs [ options ]

DESCRIPTION
  This page intentionally left blank.

SOURCE
  MIT

SEE ALSO
  sam(1), vi(1)

BUGS
  Yes.
NAME
eqn - typeset mathematics

SYNOPSIS
eqn [ option ...] [ file ...]

DESCRIPTION
Eqn is a troff(1) preprocessor for typesetting mathematics on a typesetter. Usage is almost always
eqn file ... | troff
If no files are specified, eqn reads from the standard input. Eqn prepares output for the typesetter
named in the -Tdest option (default -Tutf; see troff(1)). When run with other preprocessor filters,
eqn usually comes last.
A line beginning with .EQ marks the start of an equation; the end of an equation is marked by a
line beginning with .EN. Neither of these lines is altered, so they may be defined in macro packages
to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; text
between delimiters is also eqn input. Delimiters may be set to characters x and y with the option
-dxxy or (more commonly) with delim xy between .EQ and .EN. Left and right delimiters may be identical.
(They are customarily taken to be $$). Delimiters are turned off by delim off. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

Tokens within eqn are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes.
Braces {} are used for grouping; generally speaking, anywhere a single character like x could appear, a complicated construction enclosed in braces may be used instead. Tilde ~ represents a full space in the output, circumflex ^ half as much.

Subscripts and superscripts are produced with the keywords sub and sup. Thus x sub i makes x_i, a sub i sup 2 produces a_i^2, and e sup {x sup 2 + y sup 2} gives e^{x^2+y^2}.

Over makes fractions: a over b yields \frac{a}{b}.

Sqrt produces square roots: 1 over sqrt {ax sup 2 +bx+c} results in \frac{1}{\sqrt{ax^2+bx+c}}.

The keywords from and to introduce lower and upper limits on arbitrary things: lim_{n \rightarrow \infty} \sum_{i=0}^{n} x_i is made with lim from {n \rightarrow \infty} \sum_{i=0}^{n} x_i.

Left and right brackets, braces, etc., of the right height are made with left and right: left [ x sup 2 + y sup 2 over alpha right ] \approx \approx 1 produces \left[ \frac{x^2+y^2}{\alpha} \right] = 1. The right clause is optional. Legal characters after left and right are braces, brackets, bars, c and f for ceiling and floor, and "$" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with pile, lpile, cpile, and rpile: pile {a above b above c} produces \begin{pmatrix} a \\ b \\ c \end{pmatrix}. There can be an arbitrary number of elements in a pile. lpile left-justifies, pile and cpile center, with different vertical spacing, and rpile right-justifies.

Matrices are made with matrix: matrix \{ 1col \{ x sub i above y sub 2 \} ccol \{ 1 above 2 \} \} produces \begin{bmatrix} x_i \\ 1 \\ y_2 \end{bmatrix}. In addition, there is rcol for a right-justified column.

Diacritical marks are made with prime, dot, dotdot, hat, tildes, bar, under, vec, dyad, and under: x sub 0 sup prime = f(t) bar + g(t) under is x' = \bar{f(t)} + \underline{g(t)}, and x vec = y dyad is x = \vec{y}.

Sizes and fonts can be changed with prefix operators size n, size \pm n, fat, roman, italic, bold, or font n. Size and fonts can be changed globally in a document by gsize n and gfont n, or by the command-line arguments -s n and -f n.

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument -p n.
Successive display arguments can be lined up. Place mark before the desired lineup point in the first equation; place lineup at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with define: define thing %replacement % defines a new token called thing which will be replaced by replacement whenever it appears thereafter. The % may be any character that does not occur in replacement.

Keywords like \( \sum(\Sigma) \), \( \int(\int) \), \( \inf(\infty) \), and shorthands like \( \geq(\geq) \), \( \rightarrow(\rightarrow) \), and \( \neq(\neq) \) are recognized. Greek letters are spelled out in the desired case, as in \( \text{alpha} \) or \( \text{GAMMA} \). Mathematical words like \( \text{sin} \), \( \text{cos} \), \( \text{log} \) are made Roman automatically. \textit{Troff(1)} four-character escapes like \( \text{lh(\text{lh})} \) can be used anywhere. Strings enclosed in double quotes " " are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with \textit{troff} when all else fails.

**FILES**
/sys/lib/troff/font/devutf  font descriptions for PostScript

**SOURCE**
/sys/src/cmd/eqn

**SEE ALSO**
\texttt{troff(1)}, \texttt{tbl(1)}


**BUGS**
To embolden digits, parens, etc., it is necessary to quote them, as in \texttt{bold "12.3"}. 
NAME
at, drain, expect, pass – dialer scripting tools

SYNOPSIS
dial/at [−q][−t seconds] atcommand
dial/expect [−i][−t seconds] goodstring [ badstring... ]
dial/drain
dial/pass [−q]

DESCRIPTION
These commands are used to write telephone dialing scripts, mostly for PPP sessions. They all expect standard input and output to be connected to a communications device, e.g., a serial line to a modem. They communicate with the user using /dev/cons.

At sends atcommand to the modem prefixed with the string at. It then reads from the modem expecting an AT response. At will return success if it gets and OK of CONNECT response. Otherwise it will return the response as an error status. The options are:
−t set the timeout to seconds. The default is 300.
−q don't write to /dev/cons what is read from standard in. The default is to copy everything through.

Expect reads standard input looking for one of the strings given as arguments. Reading the first string causes a successful exit status. Reading any of the others causes an exit status equal to the string. The command also terminates on a timeout. The options are:
−t set the timeout to seconds. The default is 300.
−i ignore case when doing the matches.
−q don't write to /dev/cons what is read from standard in. The default is to copy everything through.

Pass copies input from /dev/cons to standard output. It terminates on a newline. The only flag is −q and means the same as it does for expect.

Drain discards any input waiting on standard input. It is used to sync up the stream at the start of dialing or after an error.

EXAMPLE
The following rc script dials out through a Hayes compatible modem on /dev/eia1 and lets the user type in a user name and password before starting ppp.

#!/bin/rc
device=/dev/eia1
telno=18005551212

fn initfn {
  dial/drain
  echo +++
  dial/at zh0
}

fn dialfn {
  dial/drain
  dial/at dt^$telno
  }
  
  # set up uart
  if( test −e $device^ctl ){
    echo −n b^$baud
    echo −n m1  # cts/rts flow control
    echo −n q64000 # big buffer
    echo −n nl   # nonblocking writes
    echo −n r1   # rts on
  }

```bash
    echo -n dl       # dtr on
    echo -n cl       # handup when we lose dcd
} > $dev^ctl

# get the modem's attention
while( ! initfn )
    sleep 1

# dial
while( ! dialfn )
    sleep 30

if( ! dial/expect -it 60 'username:' ){
    echo can't connect >[1=2]
    exit connect
}
dial/pass
if( ! dial/expect -it 60 'password:' ){
    echo can't connect >[1=2]
    exit connect
}
dial/pass
if( ! dial/expect -t 60 'ppp or telnet:' ){
    echo can't connect >[1=2]
    exit connect
}
    echo ppp
dial/expect -t 5 something
    echo connected >[1=2]

    # start ppp
    ip/ppp $primary -f
} < $dev > $dev
```

FILES

/rc/bin/ipconf/* example dialer scripts for ppp

SOURCE

/sys/src/cmd/dial/*.c

SEE ALSO

ppp(8), telco(4)
NAME
  faces, seemail, vwhois - mailbox interface

SYNOPSIS
  faces [-ih][-m maildir]
  seemail
  vwhois person ...

DESCRIPTION
  The faces command monitors incoming mail and displays in its window a representation of the user's mailbox using a small image for each message. The image is typically a portrait of the sender. Which image to display is determined by two directories /usr/$USER/lib/faces and /lib/faces. Entries in /usr/$USER/lib/faces take priority over those in /lib/faces. See face(6), for how these directories are organised.

  If the user is running plumber(4), faces reacts to plumb messages to the seemail port, typically from upas/Es, and is thus notified of message additions and deletions.

  Right-clicking on a message icon causes that message to be ‘plumbed’ to showmail. A typical plumb action will be to display the message, such as by the rule

  plumb start window mail -s $0

  The acme(1) mail reader listens to the showmail port automatically.

  If the user is not running plumber, faces reads the log file and right-clicking has no effect.

  If arrows are visible, clicking on them will scroll the display. Middle-clicking on the arrows scrolls to the end.

  Starting faces with the -i flag causes faces to read the messages in /mail/fs/mbox — or the mailboxes specified with the -m flag — upon startup.

  The -m option directs faces to watch for messages arriving in maildir instead of /mail/fs/mbox. Multiple -m flags may be used to watch multiple mailboxes.

  The -h flag causes a different, venerable behavior in which the window displays the history of messages received rather than the current state of the mail box. In particular, faces are not removed from the screen when messages are deleted. Also, in this mode clicking button 1 in the display will clear the window.

  Seemail is an rc(1) script that invokes faces -h.

  Vwhois tells faces to display the icons of the named persons, without sending a message.

FILES
  /mail/fs/mbox mail directory.

SOURCE
  /sys/src/cmd/faces
  /rc/bin/seemail
  /rc/bin/vwhois

SEE ALSO
  mail(1), marshal(1), nedmail(1), plumber(4), face(6), plumb(6)
NAME
factor, primes – factor a number, generate large primes

SYNOPSIS
factor [number]
primes start[finish]

DESCRIPTION
Factor prints number and its prime factors, each repeated the proper number of times. The number must be positive and less than $2^{54}$ (about $1.8 \times 10^{16}$).

If no number is given, factor reads a stream of numbers from the standard input and factors them. It exits on any input not a positive integer. Maximum running time is proportional to $\sqrt{n}$.

Primes prints the prime numbers ranging from start to finish, where start and finish are positive numbers less than $2^{56}$. If finish is missing, primes prints without end; if start is missing, it reads the starting number from the standard input.

SOURCE
/sys/src/cmd/factor.c
/sys/src/cmd/primes.c
NAME
  fedex, ups, usps - track shipments

SYNOPSIS
  fedex tracking-number
  ups tracking-number
  usps tracking-number

DESCRIPTION
  FedEx writes available shipment details for the given Federal Express 12-digit tracking-number on
the standard output. Ups is similar, but takes a United Parcel Service 18-digit tracking-number.  
Usps takes a US Post Office tracking-number.

SOURCE
  /rc/bin

BUGS
  Redesigns of the source website can break these programs.
NAME
  file - determine file type

SYNOPSIS
  file [-m][ file ... ]

DESCRIPTION
  File performs a series of tests on its argument files in an attempt to classify their contents by language or purpose. If no arguments are given, the classification is performed on standard input.

  If the -m flag is given, file outputs an appropriate MIME Content-Type specification describing the type and subtype of each file.

  The file types it looks for include directory, device file, zero-filled file, empty file, Plan 9 executable, PAC audio file, cpio archive, tex dvi file, archive symbol table, archive, rc script, sh script, PostScript, troff output file for various devices, mail box, GIF, FAX, object code, C and Alef source, assembler source, compressed files, encrypted file, English text, compressed image, image, subfont, and font.

  If a file has no apparent format, file looks at the character set it uses to classify it according to ASCII, extended ASCII, Latin ASCII, or UTF holding one or more of the following blocks of the Unicode Standard: Extended Latin, Greek, Cyrillic, Armenian, Hebrew, Arabic, Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Thai, Lao, Tibetan, Georgian, Japanese, Chinese, or Korean.

  If all else fails, file decides its input is binary.

SOURCE
  /sys/src/cmd/file.c

BUGS
  It can make mistakes.
NAME
filter, list, deliver, token, vf - filtering mail

SYNOPSIS
upas/filter [-bh] rcvr mailbox [ regexp file ] ...
upas/list [-d] add|check patternfile addressfile ...
upas/deliver recipient fromfile mbox
upas/token key [ tokenfile ]
upas/vf [-r][ -s savefile ]

DESCRIPTION
A user may filter all incoming mail by creating a world readable/executable file
/mail/box/username/pipeto. If the file is a shell script, it can use the commands
described here to implement a filter.

Filter provides simple mail filtering. The first two arguments are the recipient's address and mail-
box, that is, the same arguments provided to pipeto. The remaining arguments are all pairs of
a regular expression and a file name. With no flags, the sender's address is matched against each
regular expression starting with the first. If the expression matches, then the message is delivered
to the file whose name follows the expression. The file must be world writable and should be
append only. A message that matches none of the expressions is delivered into the user's stan-
dard mail box.

By default, filter matches each regular expression against the message's sender. The -h flag
causes filter to match against the entire header, and the -b flag causes filter to match against the
entire message (header and body).

For example, to delete any messages of precedence bulk, place in your pipeto file:

/bin/upas/filter -h $1 $2 'Precedence: bulk' /dev/null

Three other commands exist which, combined by an rc(1) script, allow you to build your own filter.
List takes two verbs; check and add. Check directs list to check each address contained in the
addressfiles against a list of patterns in patternfile. Patterns come in four forms:

~regular-expression  If any address matches the regular expression, list returns successfully.
=string.             If any address exactly matches string, list returns successfully.
!~regular-expression If any address matches the regular expression and no other address
matches a non '! ' rule, list returns error status "!match".
!=string             If any address exactly matches string and no other address matches a non
'!' rule, list returns error status "!match".

If no addresses match a pattern, list returns "no match".

The pattern file may also contain lines of the form

#include filename

to allow pattern files to include other pattern files. All pattern matches are case insensitive. List
searches the pattern file (and its includes) in order. The first matching pattern determines the
action.

List add directs list to add a pattern to patternfile for each address in the addressfiles that doesn't
already match a pattern.

Token, with only one argument, prints to standard output a unique token created from the current
date and key. With two arguments, it checks token against tokens created over the last 10 days
with key. If a match is found, it returns successfully.

Deliver delivers into mail box mbox the message read from standard input. It obeys standard mail
file locking and logging conventions.

/sys/src/cmd/upas/filterkit/pipeto.sample is a sample pipeto using the filter
kit.
A sample pipefrom, /sys/src/cmd/upas/filterkit/pipefrom.sample, is provided which adds all addresses of your outgoing mail to your pattern file. You should copy it into a directory that normally gets bound by your profile onto /bin.

Vf (virus filter) takes a mail message as standard input and searches for executable MIME attachments, either rewriting them to be non-executable or rejecting the message. The behavior depends on the attachment’s file name extension and MIME content type. /sys/lib/mimetype contains the list of known extensions and MIME content types. The fifth field of each line specifies the safety of a particular file type: y (yes), m (maybe; treated same as yes), n (no), p (previous), or r (reject). Vf allows attachments with safety y or m to pass through unaltered. Attachments with safety n both are wrapped in extra MIME headers and have .suspect appended to their file names, to avoid automatic execution by mail readers. Attachments with safety r (currently, .bat, .com, .exe, and .scr, all Microsoft executable extensions) are taken as cause for the entire message to be rejected. A safety of p (used for the x–gunzip mime type) causes the previous extension to be tested, so that x.tar.gz is treated the same as x.tar.

If /mail/lib/validateattachment exists and is executable, vf runs it on all attachments with safety n (attachments it would normally sanitize). If validateattachment’s exit status contains the string discard, vf rejects the entire message. If the status contains the string accept, vf does not sanitize the attachment. Otherwise, vf sanitizes the attachment as before. The standard validateattachment uses file(1) to determine the file type. It accepts text and image files and discards messages containing executables or zip (see gzip(1)) archives of executables.

The –r option causes vf not to sanitize MIME attachments, but instead to reject messages it determines to be viruses. The –s option causes vf to log all attachments of safety r in the mail box savefile.

FILES
/mail/box/*/pipeto mail filter
/sys/lib/mimetype MIME content types
/mail/lib/validateattachment attachment checker

SOURCE
/sys/src/cmd/upas/send
/sys/src/cmd/upas/filterkit
/sys/src/cmd/upas/vf

SEE ALSO
aliasmail(8), faces(1), mail(1), marshal(1), mlmgr(1), nedmail(1), qer(8), rewrite(6), send(8), smtp(8), upasfs(4)
NAME
fmt, htmlfmt - simple text formatters

SYNOPSIS
fmt [ option ...] [ file ...]
htmlfmt [−a] [−c charset][−u url][ file ...]

DESCRIPTION
Fmt copies the given files (standard input by default) to its standard output, filling and indenting
lines. The options are
−l n Output line length is n, including indent (default 70).
−w n A synonym for −l.
−i n Indent n spaces (default 0).
−j Do not join short lines: only fold long lines.
Empty lines and initial white space in input lines are preserved. Empty lines are inserted between
input files.
Fmt is idempotent: it leaves already formatted text unchanged.
Htmlfmt performs a similar service, but accepts as input text formatted with HTML tags. It accepts
fmt's −l and −w flags and also:
−a Normally htmlfmt suppresses the contents of form fields and anchors (URLs and image
files); this flag causes it to print them, in square brackets.
−c charset change the default character set from iso−8859−1 to charset. This is the character set
assumed if there isn’t one specified by the html itself in a <meta> directive.
−u url Use url as the base URL for the document when displaying anchors; sets −a.

SOURCE
/sys/src/cmd/fmt.c
/sys/src/cmd/htmlfmt

BUGS
Htmlfmt makes no attempt to render the two−dimensional geometry of tables; it just treats the
table entries as plain, to−be−formatted text.
NAME
   fortune - sample lines from a file

SYNOPSIS
   fortune [file]

DESCRIPTION
   Fortune prints a one-line aphorism chosen at random. If a file is specified, the saying is taken
   from that file; otherwise it is selected from /sys/games/lib/fortunes.

FILES
   /sys/games/lib/fortunes
   /sys/games/lib/fortunes.index  fast lookup table, maintained automatically

SOURCE
   /sys/src/cmd/fortune.c
NAME
freq - print histogram of character frequencies

SYNOPSIS
freq [-cdorx] [file ...]

DESCRIPTION
Freq reads the given files (default standard input) and prints histograms of the character frequencies. By default, freq counts each byte as a character; under the -r option it instead counts UTF sequences, that is, runes.

Each non-zero entry of the table is printed preceded by the byte value, in decimal, octal, hex, and Unicode character (if printable). If any options are given, the -d, -x, -o, -c flags specify a subset of value formats: decimal, hex, octal, and character, respectively.

SOURCE
/sys/src/cmd/freq.c

SEE ALSO
utf(6), wc(1)
NAME
4s, 5s, festoon, juggle, life, mahjongg, memo, sokoban, sudoku - time wasters

SYNOPSIS
games/4s
games/5s
games/festoon [-pet] [sentences [percent-invented-nouns]]
games/juggle [-d delay] [-h hands] [start] pattern
games/life startfile
games/mahjongg [-c] [-f] [-b background] [-t tileset] [-l layout]
games/memo [-h]
games/sokoban [level]
games/sudoku

DESCRIPTION
There are a few games in /bin/games:

4s, 5s  Try to fill complete rows using 4-square or 5-square tiles. Move tiles left or right by moving the mouse. Rotate tiles with buttons 1 and 3. Drop tiles for more points with button 2 or the space bar. Keys a and j move left, s and k rotate left, d and l rotate right, f and ; move right. z, p and Esc toggle suspend/resume. q, Del and control-D quit.

festoon  Generate an official-looking but utterly nonsensical bureaucratic report as pic | eqn | tbl | troff --mm input. Options --p, --e and --t add gibberish diagrams, equations and tables.

juggle  Display the juggling pattern using the optional initial start pattern. The number of hands involved (default 2) can be specified with -h, and delay can be used to speed up or slow down the action (default is 20). Try the pattern 333333441333333 or 3333535051512333333 or YWUSQOMKIGECA (see http://seehuhn.de/jong/theory.html).

life  Play the game of Life, given an initial position. There is a library of interesting initial positions; the library is consulted if startfile cannot be found.

mahjongg  Remove all tiles from the board. Click on tiles with the same face that are not blocked by others. A blocked tile is one that is partially or fully covered on top or has neighbouring tiles to the left and right. The game finishes when either all tiles are gone or there are no more moves left. The arguments are for changing background (-b), tile (-t) and layout (-l) images; -c selects a true-color buffer image, for use with drawterm or in case selecting a tile obscures it completely; -f causes mahjongg to indicate non-blocked tiles on mouse-over. The N key will generate a new level, R restarts the current one. Q and Del quit, H gives a hint, either trying to match the currently selected tile, or if no tile is selected finding out the first available tile. U and Bksp undo the last move, C tries to solve the level.

memo  Remove all tiles from the board. At first, pictures of various Bell Labs employees, Lucent Technologies' logo, and Glenda will appear. Memorize the sequence, then click to hide them and begin. Use the mouse to select two tiles. If they are the same, the tiles will disappear, otherwise the tiles will flip back and you will get a chance to try again. Button 3 generates a menu allowing you to restart, switch between easy and hard modes, and exit. The -h option sets the game to hard mode. Once the game has been completed, a message pops up with how long it took to win. Use the button 3 menu to choose a mode, or click to play again.

sokoban  Guide Glenda through a room full of walls, pebbles and holes to put the pebbles in. Your goal is to arrange all pebbles into holes by pushing them around, but you can only push a pebble if there is no wall or another pebble blocking the way. Arrow keys move Glenda up-down-left-right. N and P keys switch between the next and previous levels, R restarts the current level. Del and Q quit. Button 3 invokes a menu to restart the current level, load different level sets, and enable/disable animation of multi-step moves. Button 1 lets
you do multi-step moves and pushes, by clicking it on the destination where you want Glenda to go. Glenda will only move if it can reach the destination. For a multi-step push the pebble must be next to Glenda, the destination must be on the same row or column, and there must be a free place next to the destination where the pebble can be pushed to. Otherwise, if possible, Glenda will walk to the destination without pushing the pebble. Sokoban accepts a level file as its argument.

**Sudoku** is a puzzle game from Japan. The goal of the game is to fill the numbers 1 to 9 in all squares of the 9x9 board following a few simple rules: no digit should repeat on the same row and column, and no digit should repeat in the same 3x3 boxes outlined with thicker lines. The board is initially filled with a partial solution which can be used for inferring digits for the empty squares. The top row of the board contains the digits 1 through 9, clicking on one of those digits selects that number for placement on the board, clicking it again will deselect that digit. Clicking on an empty square will then affix the square with the selected digit or, if no digit is selected empty the square.

Button 3 presents a menu with the following options:

- **New** autogenerate a new, random board
- **Check** mark in red any digits not placed according to the rules
- **Solve** present the board's solution
- **Clear** clear the board to its starting (or last loaded) state
- **Save** save the current board to /tmp/sudoku-save
- **Load** load the last saved board from /tmp/sudoku-save
- **Print** print the current board and solution in a format suitable for addition in the sudoku library to /tmp/sudoku-board
- **Offline** pretty-print the board for off-line solving to /tmp/sudoku-print
- **Exit** quit the game

Button 2 presents a list of sudoku boards of varying degrees of difficulty from /sys/games/lib/sudoku/boards.

Pressing the Q key quits sudoku.

**FILES**

- /sys/games/lib/[45]scores score files of 4s and 5s
- /sys/games/lib/life/* interesting starting positions
- /sys/games/lib/mahjongg/* image sprites, levels and backgrounds used by mahjongg
- /lib/mc/* tiles for memo
- /sys/games/lib/sokoban/* image sprites and levels used by sokoban
- /sys/games/lib/sudoku/* images and boards used by sudoku

**SOURCE**

/sys/src/games

**BUGS**

In 4s and 5s, mouse warping (when the game is resumed, and when a new tile appears) does not happen when the mouse cursor is outside the game window. Those who prefer to use the keyboard without the mouse cursor blocking the view (or being warped all the time) may consider this a feature.
NAME
  grap - pic preprocessor for drawing graphs

SYNOPSIS
  grap [ file ... ]

DESCRIPTION
Grap is a pic(1) preprocessor for drawing graphs on a typesetter. Graphs are surrounded by the
 troff 'commands' .G1 and .G2. Data are scaled and plotted, with tick marks supplied automatically. Commands exist to modify the frame, add labels, override the default ticks, change the plotting style, define coordinate ranges and transformations, and include data from files. In addition, grap provides the same loops, conditional, and macro processing that pic does.

frame ht e wid e top dotted ...: Set the frame around the graph to specified ht and wid, default is 2 by 3 (inches). The line styles (dotted, dashed, invis, solid (default)) of the sides (top, bot, left, right) of the frame can be set independently.

label side "a label" "as a set of strings" adjust: Place label on specified side; default side is bottom. adjust is up (or down left right) exp in shift default position; width expr sets the width explicitly.

ticks side in at optname expr, expr, ...: Put ticks on side at expr, ..., and label with "expr". If any expr is followed by "...", label tick with "...", and turn off all automatic labels. If "..." contains %f's, they will be interpreted as printf formatting instructions for the tick value. Ticks point in or out (default out). Tick iterator: instead of at ..., use from expr to expr by op expr where op is optionally +−*/ for additive or multiplicative steps. by can be omitted, to give steps of size 1. If no ticks are requested, they are supplied automatically; suppress this with ticks off. Automatic ticks normally leave a margin of 7% on each side; set this to anything by margin = expr.

grid side linedesc at optname expr, expr, ...: Draw grids perpendicular to side in style linedesc at expr, ... Iterators and labels work as with ticks.

coord optname x min, max y min, max log x log y: Set range of coords and optional log scaling on either or both. This overrides computation of data range. Default value of optname is current coordinate system (each coord defines a new coordinate system).

plot "str" at point; "str" at point: Put str at point. Text position can be qualified with rjust, ljust, above, below after "...".

ticks side at point: Draw line from here to there. arrow works in place of line.

next optname at point linedesc: Continue plot of data in optname to point; default is current.

draw optname linedesc ...: Set mode for next: use this style from now on, and plot "..." at each point (if given).

new optname linedesc ...: Set mode for next, but disconnect from previous.

A list of numbers x y1 y2 y3 ... is treated as plot bullet at x,y1; plot bullet at x,y2; etc., or as next at x,y1 etc., if draw is specified. Abscissae of 1,2,3,... are provided if there is only one input number per line.

A point optname expr, expr maps the point to the named coordinate system. A linedesc is one of dot dash invis solid optionally followed by an expression.

define name { whatever}: Define a macro. There are macros already defined for standard plotting symbols like bullet, circle, star, plus, etc., in /sys/lib/grap.defines, which is included if it exists.

var = expr: Evaluate an expression. Operators are + − * and /. Functions are log and exp (both base 10), sin, cos, sqrt; rand returns random number on [0,1); max(e,e), min(e,e), int(e).

print expr; print "...": As a debugging aid, print expr or string on the standard error.

copy "file name": Include this file right here.
copy thru macro: Pass rest of input (until .G2) through macro, treating each field (non-blank, or ".") as an argument. macro can be the name of a macro previously defined, or the body of one in place, like /plot $1 at $2; $3/.
copy thru macro until ".string": Stop copy when input is string (left-justified).
pic remainder of line: Copy to output with leading blanks removed.
graph Name pic-position: Start a new frame, place it at specified position, e.g., graph Thing2
with .sw at Thing1.se + (0.1,0). Name must be capitalized to keep pic happy.
   . anything at beginning of line: Copied verbatim.
sh %anything %: Pass everything between the %’s to the shell; as with macros, % may be any char-
   acter and anything may include newlines.
   # anything: A comment, which is discarded.

Order is mostly irrelevant; no category is mandatory. Any arguments on the .G1 line are placed
on the generated .PS line for pic.

EXAMPLES
.G1
.frame ht 1 top invis right invis
coord x 0, 10 y 1, 3 log y
ticks left in at 1 "bottommost tick", 2,3 "top tick"
ticks bot in from 0 to 10 by 2
label bot "silly graph"
label left "left side label" "here"
grid left dashed at 2.5
copy thru / circle at $1,$2 /
 1 1
 2 1.5
 3 2
 4 1.5
10 3
.G2

silly graph

FILES
/sys/lib/grap.defines definitions of standard plotting characters, e.g., bullet

SOURCE
/sys/src/cmd/grap

SEE ALSO
pic(1), troff(1)
J. L. Bentley and B. W. Kernighan, “GRAP—A Language for Typesetting Graphs”, Unix Research Sys-
NAME
graph - draw a graph

SYNOPSIS
graph [ option ...]

DESCRIPTION
Graph with no options takes pairs of numbers from the standard input as abscissas (x-values) and
ordinates (y-values) of a graph. Successive points are connected by straight lines. The graph is
encoded on the standard output for display by plot(1) filters.

If an ordinate is followed by a nonnumeric string, that string is printed as a label beginning on the
point. Labels may be surrounded with quotes " " in which case they may be empty or contain
blanks and numbers; labels never contain newlines.

The following options are recognized, each as a separate argument.

-a Supply abscissas automatically; no x-values appear in the input. Spacing is given by the
next argument (default 1). A second optional argument is the starting point for automatic
abscissas (default 0, or 1 with a log scale in x, or the lower limit given by –x).

-b Break (disconnect) the graph after each label in the input.

-c Character string given by next argument is default label for each point.

-g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

-l Next argument is a legend to title the graph. Grid ranges are automatically printed as part
of the title unless a –s option is present.

-m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected. Some
devices give distinguishable line styles for other small integers. Mode –1 (default) begins
with style 1 and rotates styles for successive curves under option –o.

-o (Overlay.) The ordinates for n superposed curves appear in the input with each abscissa
value. The next argument is n.

-p Next argument is one or more of the characters bcgkmyw, choosing pen colors by their
initial letter, as in plot(6). Successive curves will cycle through the colors in the given order.

-s Save screen; no new page for this graph.

-x 1 If 1 is present, x-axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x lim-
its. Third argument, if present, is grid spacing on x axis. Normally these quantities are
determined automatically.

-y 1 Similarly for y.

-e Make automatically determined x and y scales equal.

-h Next argument is fraction of space for height.

-w Similarly for width.

-r Next argument is fraction of space to move right before plotting.

-u Similarly to move up before plotting.

-t Transpose horizontal and vertical axes. (Option –a now applies to the vertical axis.)

If a specified lower limit exceeds the upper limit, the axis is reversed.

SOURCE
/sys/src/cmd/graph

SEE ALSO
plot(1), grap(1)

BUGS
Segments that run out of bounds are dropped, not windowed. Logarithmic axes may not be
reversed. Option –e actually makes automatic limits, rather than automatic scaling, equal.
NAME
grep - search a file for a pattern

SYNOPSIS
grep [-bchllns][ -e ] pattern | -E patternfile [ file ... ]

DESCRIPTION
Grep searches the input files (standard input default) for lines that match the pattern, a regular expression as defined in regexp(6) with the addition of a newline character as an alternative (substitute for \) with lowest precedence. Normally, each line matching the pattern is 'selected', and each selected line is copied to the standard output. The options are

- c  Print only a count of matching lines.
- h  Do not print file name tags (headers) with output lines.
- e  The following argument is taken as a pattern. This option makes it easy to specify patterns that might confuse argument parsing, such as --n.
- i  Ignore alphabetic case distinctions. The implementation folds into lower case all letters in the pattern and input before interpretation. Matched lines are printed in their original form.
- l  (ell) Print the names of files with selected lines; don't print the lines.
- L  Print the names of files with no selected lines; the converse of --l.
- n  Mark each printed line with its line number counted in its file.
- s  Produce no output, but return status.
- v  Reverse: print lines that do not match the pattern.
- f  The pattern argument is the name of a file containing regular expressions one per line.
- b  Don't buffer the output: write each output line as soon as it is discovered.

Output lines are tagged by file name when there is more than one input file. (To force this tagging, include /dev/null as a file name argument.)

Care should be taken when using the shell metacharacters $*[^|()=\ and newline in pattern; it is safest to enclose the entire expression in single quotes ' ... ' . An expression starting with 's' will treat the rest of the expression as literal characters.

SOURCE
/sys/src/cmd/grep

SEE ALSO
ed(1), awk(1), sed(1), sam(1), regexp(6)

DIAGNOSTICS
Exit status is null if any lines are selected, or non-null when no lines are selected or an error occurs.
NAME
  gs - Aladdin Ghostscript (PostScript and PDF language interpreter)

SYNOPSIS
  gs [ options ] [ files ] ...

DESCRIPTION
  Ghostscript is a programming language similar to Adobe Systems’ PostScript
  and PDF languages, which are in turn similar to Forth. Gs reads files in
  sequence and executes them as Ghostscript programs. After doing this, it
  reads further input from the standard input. If the file – is named,
  however, it represents the standard input, which is read in order and not
  after the files on the command line. Each line is interpreted separately. The
  ‘quit’ command, or end-of-file, exits the interpreter.

  The interpreter recognizes several switches described below, which may appear
  anywhere in the command line and apply to all files thereafter.

  The –h or –? options give help and list the available devices; the default is
  plan9, which produces compressed image files suitable for viewing with
  page(1) (but note that page(1) will invoke gs automatically; see its manual).

  Ghostscript may be built with multiple output devices. Ghostscript normally
  opens the first one and directs output to it. To use device xyz as the initial
  output device, include the switch

    −sDEVICE=xyz

  in the command line. This switch must precede the first PostScript file and
  only its first invocation has any effect. Output devices can also be selected by
  the word selectdevice in the input language, or by setting the environment
  variable GS_DEVICE. The order of precedence for these alternatives, highest to
  lowest, is:

    selectdevice
    (command line)
    GS_DEVICE
    plan9

  Normally, output goes directly to a scratch file. To send the output to a series
  of files foo1.xyz, foo2.xyz, etc., use the switch

    −sOutputFile=foo%d.xyz

  The %d may be any printf (see fprintf(2)) format specification. Each file will
  receive one page of output. If the file name begins with a pipe character, the
  output will be sent as standard input to the following pipeline. For example,

    −sOutputFile=|lp

  Specifying the file – will send the files to standard output; this also requires
  enabling the −q option.

Initialization files
  When looking for the initialization files (gs_.ps), the files related to fonts, or
  the file for the run operator, Ghostscript first looks for the file (if it doesn’t
  start with a slash) in the current directory, then in these directories in the
  following order:

  1. Any directories specified by −I switches in the command line (see below);
  2. Any directories specified by the GS_LIB environment variable;
  3. The directories /sys/lib/ghostscript, /sys/lib/ghostscript/font, and
     /sys/lib/postscript/font.

  The GS_LIB or −I parameters may be a single directory or a colon–separated list.

Options
  −− filename arg1 ...

    Take the next argument as a file name as usual, but take all remaining
    arguments (even if they have the syntactic form of switches) and define the name
    ARGUMENTS in userdict (not systemdict) as an array of those strings, before
    running the file. When Ghostscript finishes

    −Dname=token
-d name=token
  Define a name in systemdict with the given definition. The token must be exactly one
token (as defined by the 'token' operator) and must not contain any white space.

-d name
-d name
  Define a name in systemdict with value=null.

-S name=string
-s name=string
  Define a name in systemdict with a given string as value. This is different from -d. For
example, -d name=35 is equivalent to the program fragment
/name 35 def
whereas -s name=35 is equivalent to
/name (35) def

-q Quiet startup: suppress normal startup messages, and also do the equivalent of -dQUIET.

-g number1 x number2
  Equivalent to -dDEVICEWIDTH=number1 and -dDEVICEHEIGHT=number2. This is
for the benefit of devices, such as windows, that allow width and height to be specified.

-r number
-r number1 x number2
  Equivalent to -dDEVICEXRESOLUTION=number1 and -dDEVICEYRESOLUTION=number2. This is for the benefit of devices, such as printers, that support multiple X and Y
resolutions. If only one number is given, it is used for both X and Y resolutions.

-I directories
  Adds the designated list of directories at the head of the search path for library files.
Note that gs_init.ps makes systemdict read-only, so the values of names defined with -D/d/S/s
cannot be changed (although, of course, they can be superseded by definitions in userdict or other
dictionaries.)

Special names
-d BATCH
  Exit after the last file has been processed. This is equivalent to listing quit.ps at the end of
the list of files.

-dDISKFONTS
  Causes individual character outlines to be loaded from the disk the first time they are
encountered. (Normally Ghostscript loads all the character outlines when it loads a font.)
This may allow loading more fonts into RAM, at the expense of slower rendering.

-dNOCACHE
  Disables character caching. Only useful for debugging.

-dNOBIND
  Disables the 'bind' operator. Only useful for debugging.

-dNODISPLAY
  Suppresses the normal initialization of the output device. This may be useful when debug-
ging.

-dNOPAUSE
  Disables the prompt and pause at the end of each page. This may be desirable for applica-
tions where another program (e.g. page1) is 'driving' Ghostscript.

-dSAFER
  Disables the deletefile and renamefile operators, and the ability to open files in
any mode other than read-only. This may be desirable for spoolers or other sensitive environ-
ments. Files in the /Ed directory may still be opened for writing.

-dWRITESYSTEMDICT
  Leaves systemdict writable. This is necessary when running special utility programs such
as font2c and pcharstr, which must bypass normal PostScript access protection.


-sDEVICE=\textit{device}
Selects an alternate initial output device, as described above.

-sOutputFile=\textit{filename}
Selects an alternate output file (or pipe) for the initial output device, as described above.

\textbf{FILES}
/sys/lib/ghostscript/*
Startup-files, utilities, examples, and basic font definitions.
/sys/lib/ghostscript/fonts/*
Additional font definitions.

\textbf{SOURCE}
/sys/src/cmd/gs

\textbf{SEE ALSO}
\textit{page}(1), \textit{ps2pdf}(1)
The Ghostscript document files in \texttt{doc} and \texttt{man} subdirectories of the source directory.

\textbf{BUGS}
The treatment of standard input is non-standard.
NAME
gview – interactive graph viewer

SYNOPSIS
gview [−mp][−l logfile][files]

DESCRIPTION
gview reads polygonal lines or a polygonal line drawing from an ASCII input file (which defaults to standard input), and views it interactively, with commands to zoom in and out, perform simple editing operations, and display information about points and polylines. (Multiple input files are allowed if you want to overlay several line drawings.) The editing commands can change the color and thickness of the polylines, delete (or undelete) some of them, and optionally rotate and move them. It is also possible to generate an output file that reflects these changes and is in the same format as the input.

Since the move and rotate commands are undesirable when just viewing a graph, they are only enabled if gview is invoked with the −m option.

The −p option plots only the vertices of the polygons.

Clicking on a polyline with button 1 displays the coordinates and a t value that tells how far along the polyline. (t=0 at the first vertex, t=1 at the first vertex, t=1.5 halfway between the second and third vertices, etc.) The −l option generates a log file that lists all points selected in this manner.

The most important interactive operations are to zoom in by sweeping out a rectangle, or to zoom out so that everything currently being displayed shrinks to fit in the swept-out rectangle. Other options on the button 3 menu are unzoom which restores the coordinate system to the default state where everything fits on the screen, recenter which takes a point and makes it the center of the window, and square up which makes the horizontal and vertical scale factors equal.

To take a graph of a function where some part is almost linear and see how it deviates from a straight line, select two points on this part of the graph (i.e., select one with button 1 and then select the other) and then use the slant command on the button 3 menu. This slants the coordinate system so that the line between the two selected points appears horizontal (but vertical still means positive y). Then the zoom in command can be used to accentuate deviations from horizontal. There is also an unslant command that undoes all of this and goes back to an unslanted coordinate system.

There is a recolor command on button 3 that lets you select a color and change everything to have that color, and a similar command on button 2 that only affects the selected polyline. If the input file uses the Multi(...) feature explained below, either flavor of recolor allows you to type a digit in lieu of selecting a color.

The thick or thin command on button 2 changes the thickness of the selected polyline and there is also an undo command for such edits. Finally, button 3 has commands to read a new input file and display it on top of everything else, restack the drawing order (in case lines of different color are drawn on top of each other), write everything into an output file, or exit the program.

Each polyline in an input or output file is a space–delimited x y coordinate pair on a line by itself, and the polyline is a sequence of such vertices followed by a label. The label could be just a blank line or it could be a string in double quotes, or virtually any text that does not contain spaces and is on a line by itself. The label at the end of the last polyline is optional. It is not legal to have two consecutive labels, since that would denote a zero–vertex polyline and each polyline must have at least one vertex. (One-vertex polylines are useful for scatter plots.) Under the −l option, a newline causes the selected polyline’s label to appear in the log file (where it could be seen by invoking tail −f in another window).

If the label after a polyline contains the word Thick or a color name (Red, Pink, Dkred, Orange, Yellow, Dkyellow, Green, Dkgreen, Cyan, Blue, Ltblue, Magenta, Violet, Gray, Black, White), whichever color name comes first will be used to color the polyline. Alternatively, labels can contain Multi followed by single–letter versions of these names: (R, P, r, O, Y, y, G, g, C, B, b, M, V, A, K, W, each optionally preceded by T). Then recolor followed by a nonzero digit n selects the nth alternative for each polyline.
EXAMPLE
To see a graph of the function \( y = \frac{\sin(x)}{x} \) generate input with an awk script and pipe it into gview:

```bash
awk 'BEGIN{for(x=.1;x<500;x+=.1)print x, sin(x)/x}' | gview
```

SOURCE
/sys/src/cmd/gview.c

SEE ALSO
awk(1), tail(1)

BUGS
The user interface for the slant command is counter-intuitive. Perhaps it would be better to have a scheme for sweeping out a parallelogram.

The \(-p\) option makes the interactive point selection feature behave strangely, and is unnecessary since extra blank lines in the input achieve essentially the same effect.
NAME
gzip, gunzip, bzip2, bunzip2, compress, uncompress, zip, unzip - compress and expand data

SYNOPSIS
gzip [-cvD[1-9]] [file ...]
gunzip [-ctTvD] [file ...]
bzip2 [-cvD[1-9]] [file ...]
bunzip2 [-cvD] [file ...]
compress [-cv] [file ...]
uncompress [-cv] [file ...]
zip [-avD[1-9]] [-f zipfile] [file ...]
unzip [-cistTvD] [-f zipfile] [file ...]

DESCRIPTION
Gzip encodes files with a hybrid Lempel-Ziv 1977 and Huffman compression algorithm known as deflate. Most of the time, the resulting file is smaller, and will never be much bigger. Output files are named by taking the last path element of each file argument and appending .gz; if the resulting name ends with .tar.gz, it is converted to .tgz instead. Gunzip reverses the process. Its output files are named by taking the last path element of each file argument, converting .tgz to .tar.gz, and stripping any .gz; the resulting name must be different from the original name.

Bzip2 and bunzip2 are similar in interface to gzip and gunzip, but use a modified Burrows-Wheeler block sorting compression algorithm. The default suffix for output files is .tbz2, with .tar.bz2 becoming .tbz. Bunzip2 recognizes the extension .tbz2 as a synonym for .tbz.

Compress and uncompress are similar in interface to gzip and gunzip, but use the Lempel-Ziv-Welch compression algorithm. The default suffix for output files is .Z. Compress is one of the oldest widespread Unix compression programs.

Zip encodes the named files and places the results into the archive zipfile, or the standard output if no file is given. Unzip extracts files from an archive created by zip. If no files are named as arguments, all of files in the archive are extracted. A directory's name implies all recursively contained files and subdirectories. Zip is the de facto standard for compression on Microsoft operating systems.

None of these programs removes the original files. If the process fails, the faulty output files are removed.

The options are:
- -a Automaticaly creates directories as needed, needed for zip files created by broken implementations which omit directories.
- -c Write to standard output rather than creating an output file.
- -i Convert all archive file names to lower case.
- -s Streaming mode. Looks at the file data adjacent to each compressed file rather than seeking in the central file directory. This is the mode used by unzip if no zipfile is specified. If -s is given, -T is ignored.
- -t List matching files in the archive rather than extracting them.
- -T Set the output time to that specified in the archive.
- -1 .. -9 Sets the compression level. -1 is tuned for speed, -9 for minimal output size. The best compromise is -6, the default.
- -v Produce more descriptive output. With -t, adds the uncompressed size in bytes and the modification time to the output. Without -t, prints the names of files on standard error as they are compressed or decompressed.
- -D Produce debugging output.
SOURCE
/sys/src/cmd/gzip
/sys/src/cmd/bzip2
/sys/src/cmd/compress

SEE ALSO
  tar(1)

BUGS
Unzip can only extract files which are uncompressed or compressed with the deflate compression scheme. Recent zip files fall into this category. Very recent zip files may have tables of contents that unzip cannot read. Such files are still readable by invoking unzip with the −s option.
NAME
hget - retrieve a web page corresponding to a url

SYNOPSIS
hget [ −dhv ] [ −o ofile ] [ −p body ] [ −x netmntpt ] [ −x header ] url

DESCRIPTION
Hget retrieves the web page specified by the URL url and writes it, absent the −o option, to standard output. The known URL types are: http and ftp.

If url is of type HTTP and the −p option is specified, then an HTTP POST is performed with body as the data to be posted.

The −o option is used to keep a local file in sync with a web page. If the web page has been modified later than the file, it is copied into the file. If the file is up to date but incomplete, hget will fetch the missing bytes.

Option −h causes HTTP headers to be printed to standard output in addition to the transferred web page.

Option −x sends an arbitrary HTTP header.

Option −d turns on debugging written to standard error.

Normally, hget uses the IP stack mounted under /net. The −x option can be used to specify the mount point of a different IP stack to use.

Option −v writes progress lines to standard error once a second. Each line contains two numbers, the bytes transferred so far and the total length to be transferred.

If the environment variable httpproxy is set, it is used as a URL denoting an HTTP proxy server. All HTTP accesses use this server to get the page instead of calling the destination server.

SOURCE
/sys/src/cmd/hget.c

SEE ALSO
ftpfs(4)
NAME
 history - print file names from the dump

SYNOPSIS
 history [-Dabcemnw] [-fuv] [-d dumpfilesystem] [-s yyyymmdd] files ...

DESCRIPTION
 History prints the names, dates, and sizes, and modifier of all versions of the named files, looking backwards in time, stored in the dump file system. If the file exists in the main tree, the first line of output will be its current state. For example,

    history /adm/users

produces

    May 14 15:29:18 EDT 2001 /adm/users 10083 [adm]
    May 11 17:26:24 EDT 2001 /n/dump/2001/0514/adm/users 10481 [adm]
    May 10 16:40:51 EDT 2001 /n/dump/2001/0511/adm/users 10476 [adm]
    ...

When presented with a path of the form /n/ fs/ path, history will use fsdump as the name of the dump file system, and will print a history of path.

The -v option enables verbose debugging printout.

The -D option causes diff(1) to be run for each adjacent pair of dump files. The options -abcemnw are passed through to diff; the little-used diff option -e is replaced by the functionality described below, and the -r option is disallowed.

The -u option causes times to be printed in GMT (UT) rather than local time.

The -d option selects some other dump file system such as /n/bootesdump.

The -f option forces the search to continue even when the file in question does not exist (useful for files that only exist intermittently).

Finally, the -s option sets the starting (most recent) date for the output.

EXAMPLES
 Check how often a user has been logged in.

    history /usr/ches/tmp

FILES
 /n/dump

SOURCE
 /sys/src/cmd/history.c

SEE ALSO
 fs(4)
 yesterday(1)
NAME
hoc - interactive floating point language
SYNOPSIS
hoc [ –e expression ] [ file ... ]
DESCRIPTION
Hoc interprets a simple language for floating point arithmetic, at about the level of BASIC, with C-
like syntax and functions.
The named files are read and interpreted in order. If no file is given or if file is – hoc interprets the
standard input. The –e option allows input to hoc to be specified on the command line, to be
treated as if it appeared in a file.
Hoc input consists of expressions and statements. Expressions are evaluated and their results
printed. Statements, typically assignments and function or procedure definitions, produce no output
unless they explicitly call print.
Variable names have the usual syntax, including _; the name _ by itself contains the value of the
last expression evaluated. The variables E, PI, PHI, GAMMA and DEG are predefined; the last is
59.25..., degrees per radian.
Expressions are formed with these C-like operators, listed by decreasing precedence.
^      exponentiation
!      ++      --
*      /      %
+      -
>      >=      <      <=      ==      !=
&&    
||    
=      +=      -=      *=      /=      %=
Built in functions are abs, acos, asin, atan (one argument), cos, cosh, exp, int, log,
log10, sin, sinh, sqrt, tan, and tanh. The function read(x) reads a value into the var-
able x and returns 0 at EOF; the statement print prints a list of expressions that may include
string constants such as "hello\n".
Control flow statements are if-else, while, and for, with braces for grouping. Newline ends
a statement. Backslash-newline is equivalent to a space.
Functions and procedures are introduced by the words func and proc; return is used to
return with a value from a function.
EXAMPLES
func gcd(a, b) {
    temp = abs(a) % abs(b)
    if(temp == 0) return abs(b)
    return gcd(b, temp)
}
for(i=1; i<12; i++) print gcd(i,12)
SOURCE
/sys/src/cmd/hoc
SEE ALSO
bc(1), dc(1)
BUGS
Error recovery is imperfect within function and procedure definitions.
NAME
htmlroff - HTML formatting and typesetting

SYNOPSIS
htmlroff [−iuv][−m name][−r aN][ file ... ]

DESCRIPTION
Htmlroff accepts troff(1) input in the named files and formats it as HTML for viewing in a web browser.
If no file argument is given, htmlroff reads the standard input. An argument consisting of a single
minus (−) is taken to be a file name corresponding to the standard input. The options are:
−i Read standard input after the input files are exhausted.
−m name
Process the macro file /sys/lib/tmac/tmac . name before the input files.
−r aN Set register a (one character name) to N.
−u Generate UTF output. By default, htmlroff converts Unicode runes into the corresponding
HTML entity sequences (&alpha;, &nbsp;, and so on). Htmlroff invokes tcs(1) for the
conversion.
−v Generate debugging output and warnings about suspicious input.
Most troff input files, especially those using the ms(6) macros, can be used unaltered. In general,
the macro file tmac . html should be processed after processing other standard macro files, as in
htmlroff −ms −mhtml.
Htmlroff(6) describes the changes to the input language.
Mhtml(6) describes the new macros.

EXAMPLES
Format the Plan 9 web page:
    cd /usr/web/plan9
    htmlroff −mhtml index.tr >index.html
Format a paper:
    cd /sys/doc
    pic auth . ms | tbl | eqn | htmlroff −ms −mhtml >auth . html

FILES
/sys/lib/troff/font/devutf/utfmap
Mapping from troff two-character names like \(*a to Unicode characters like α.

SOURCE
/sys/src/cmd/htmlroff

SEE ALSO
 tcs(1), troff(1), htmlroff(6), mhtml(6)
NAME
idiff - interactive diff

SYNOPSIS
idiff [−bw] file1 file2

DESCRIPTION
Idiff interactively merges file1 and file2 onto standard output. Wherever file1 and file2 differ, idiff displays the differences in the style of “diff −n” on standard error and prompts the user to select a chunk. Valid responses are:

< Use the chunk from file1.
> Use the chunk from file2.
= Use the diff output itself.
q<, q>, q=
Use the given response for all future questions.
! cmd Execute cmd and prompt again.
Idiff invokes diff(1) to compare the files. The −b and −w flags, if passed, are passed to diff.

FILES
/tmp/idiff.*

SOURCE
/sys/src/cmd/idiff.c

SEE ALSO
diff(1)
NAME
join - relational database operator

SYNOPSIS
join [options] file1 file2

DESCRIPTION
Join forms, on the standard output, a join of the two relations specified by the lines of file1 and file2. If one of the file names is -, the standard input is used.

File1 and file2 must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in file1 and file2 that have identical join fields. The output line normally consists of the common field, then the rest of the line from file1, then the rest of the line from file2.

Input fields are normally separated spaces or tabs; output fields by space. In this case, multiple separators count as one, and leading separators are discarded.

The following options are recognized, with POSIX syntax.

-a n In addition to the normal output, produce a line for each unpairable line in file n, where n is 1 or 2.

-v n Like -a, omitting output for paired lines.

-e s Replace empty output fields by string s.

-1 m
-2 m Join on the mth field of file1 or file2.

-j n m
Archaic equivalent for -n m.

-ofields
Each output line comprises the designated fields. The comma-separated field designators are either 0, meaning the join field, or have the form n.m, where n is a file number and m is a field number. Archaic usage allows separate arguments for field designators.

-tc Use character c as the only separator (tab character) on input and output. Every appearance of c in a line is significant.

EXAMPLES
sort -t: +1 /adm/users | join -t: -1 2 -a 1 -e "" - bdays
Add birthdays to the /adm/users file, leaving unknown birthdays empty. The layout of /adm/users is given in users(6); bdays contains sorted lines like ken:Feb 4, 1953.

tr : ' ' </adm/users | sort -k 3 3 >temp
join -1 3 -2 3 -o 1.1,2.1 temp temp | awk '$1 < $2'
Print all pairs of users with identical userids.

SOURCE
/sys/src/cmd/join.c

SEE ALSO
sort(1), comm(1), awk(1)

BUGS
With default field separation, the collating sequence is that of sort -b -ky,y; with -t, the sequence is that of sort -tx -ky,y.
One of the files must be randomly accessible.
NAME
    jpg, gif, png, ppm, bmp, v210, yuv, ico, togif, toppm, topng, toico – view and convert pictures

SYNOPSIS
    jpg [-39cdefFkJrtv] [file ...]
    gif [-39cdefkrtv] [file ...]
    png [-39cdefkrtv] [file ...]
    ppm [-39cdefkrtv] [file ...]
    bmp [file]
    v210 [-39cdefkrtv] [file ...]
    yuv [file]
    togif [-c comment] [-l loopcount] [-d msec] [-t transindex] [file ...] [-d msec] [file ...]
    toppm [-c comment] [file]
    topng [-c comment] [ [ -g gamma ] [file]
    ico [file]
    toico [file ...]

DESCRIPTION
    These programs read, display, and write image files in public formats. *Jpg*, *gif*, *png*, *ppm*,
    *bmp*, *v210*, and *yuv* read files in the corresponding formats and, by default, display them in the current
    window; options cause them instead to convert the images to Plan 9 image format and write them
to standard output. *Togif*, *Toppm*, and *topng* read Plan 9 images files, convert them to GIF, PPM,
or PNG, and write them to standard output.

    The default behavior of *jpg*, *gif*, and *ppm* is to display the *file*, or standard input if no file is
    named. Once a file is displayed, typing a character causes the program to display the next image.
    Typing a q, DEL, or control-D exits the program. For a more user-friendly interface, use *page*(),
    which invokes these programs to convert the images to standard format, displays them, and offers
    scrolling, panning, and menu-driven navigation among the files.

    These programs share many options:
    
    -e Disable Floyd–Steinberg error diffusion, which is used to improve the appearance of images
      on color-mapped displays, typically with 8 bits per pixel. Primarily useful for debugging; if
      the display has true RGB color, the image will be displayed in full glory.
    
    -k Convert and display the image as a black and white (really grey-scale) image.
    
    -v Convert the image to an RGBV color-mapped image, even if the display has true RGB color.
    
    -d Suppress display of the image; this is set automatically by any of the following options:
    
    -c Convert the image to a Plan 9 representation, as defined by *image*(6), and write it to stan-
      dard output.
    
    -9 Like –c, but produce an uncompressed image. This saves processing time, particularly
      when the output is being piped to another program such as *page*(), since it avoids com-
     pression and decompression.
    
    -t Convert the image, if it is in color, to a true color RGB image.
    
    -3 Like –t, but force the image to RGB even if it is originally grey-scale.

    *Jpg* has two extra options used to process the output of the LML video card:
    
    -f Merge two adjacent images, which represent the two fields of a video picture, into a single
      image.
    
    -F The input is a motion JPEG file, with multiple images representing frames of the movie.
      Sets –f.

    The *togif* and *toppm* programs go the other way: they convert from Plan 9 images to GIF and PPM,
    and have no display capability. Both accept an option –c to set the comment field of the resulting
    file. If there is only one input picture, *togif* converts the image to GIF format. If there are many
    files, though, it will assemble them into an animated GIF file. The options control this process:
−l loopcount
   By default, the animation will loop forever; loopcount specifies how many times to loop. A value of zero means loop forever and a negative value means to stop after playing the sequence once.

−d msec
   By default, the images are displayed as fast as they can be rendered. This option specifies the time, in milliseconds, to pause while displaying the next named file.

Gif translates files that contain a 'transparency' index by attaching an alpha channel to the converted image.

Ico displays a Windows icon (.ico) file. If no file is specified, ico reads from standard input. Icon files contain sets of icons represented by an image and a mask. Clicking the right button pops up a menu that lets you write any icon's image as a Plan 9 image (widthxheight.image), write any icon's mask as a Plan 9 image (widthxheight.mask), or exit. Selecting one of the write menu items yields a sight cursor. Move the sight over the icon and right click again to write.

Toico takes a list of Plan 9 image files (or standard input) and creates a single icon file. The masks in the icon file will be the white space in the image. The icon file is written to standard output.

SOURCE
/sys/src/cmd/jpg

SEE ALSO
page(1), image(6),
http://www.w3.org/Graphics/JPEG/itu-t81.pdf
http://www.w3.org/Graphics/GIF/spec-gif89a.txt
http://www.w3.org/TR/2003/REC-PNG-20031110
http://netpbm.sourceforge.net/doc/ppm.html
http://en.wikipedia.org/wiki/Windows_bitmap
http://en.wikipedia.org/wiki/Yuv

BUGS
  Writing an animated GIF using togif is a clumsy undertaking.
NAME
   kbmap - show a list of available keyboard maps and switch between them.

SYNOPSIS
   kbmap [ file... ]

DESCRIPTION
   *Kbmap* shows a single column consisting of the names of keyboard maps for different alphabets available on the system. With no arguments *kbmap* will look for files in */sys/lib/kbmap*.

   Clicking the right mouse button will highlight the entry and force the keyboard mapping defined in the corresponding file to become current for the system; typing 'q' quits.

   *Kbmap* requires that the file */dev/kbmap* served by *kbmap*(3) exists and is writable.

SOURCE
   */sys/src/cmd/kbmap.c*

SEE ALSO
   *kbmap*(3)

BUGS
   Not all keyboards map the entire set of characters, so one has to switch back to the default map before changing to another.
NAME
  kill, slay, broke - print commands to kill processes

SYNOPSIS
  kill name ...
  slay name ...
  broke [ user ]

DESCRIPTION
  Kill prints commands that will cause all processes called name and owned by the current user to be terminated. Use the send command of rio(1), or pipe the output of kill into rc(1) to execute the commands.

  Kill suggests sending a kill note to the process; the same message delivered to the process's ctl file (see proc(3)) is a surer, if heavy handed, kill, but is necessary if the offending process is ignoring notes. The slay command prints commands to do this.

  Broke prints commands that will cause all processes in the Broken state and owned by user (by default, the current user) to go away. When a process dies because of an error caught by the system, it may linger in the Broken state to allow examination with a debugger. Executing the commands printed by broke lets the system reclaim the resources used by the broken processes.

SOURCE
  /rc/bin/kill
  /rc/bin/broke

SEE ALSO
  ps(1), stop(1), notify(2), proc(3)
NAME
ktrace - interpret kernel stack dumps

SYNOPSIS
ktrace [−i] kernel pc sp [ link]

DESCRIPTION
Ktrace translates a hexadecimal kernel stack dump into a sequence of acid(1) commands to show
the points in the call trace. The kernel argument should be the path of the kernel being debugged,
and pc and sp are the PC and SP values given in the stack dump. For MIPS kernels, the contents of
the link register must also be supplied.

A stack trace consists of a ktrace command followed by a series of lines containing fields of the
form location=contents:
ktrace /kernel/path 80105bc1 8048e174
8048e114=80105ac6 8048e120=80140bb4 8048e134=8010031c
8048e16c=80137e45 8048e170=80105bc1 8048e178=80137e62
...

The trace can be edited to provide the correct kernel path and then pasted into a shell window. If
the −i option is present, ktrace instead prompts for the contents of the memory locations in
which it is interested; this is useful when the stack trace is on a screen rather than in a machine
readable form.

SOURCE
/sys/src/cmd/ktrace.c

SEE ALSO
acid(1), rdbfs(4)

BUGS
When examining a kernel trace resulting from an interrupt on top of other interrupts, only the top-
most call trace is printed.
NAME
leak, kmem, umem - help find memory leaks

SYNOPSIS
leak [-abcds] [-f binary] [-r res] [-x width] pid ...
kmem [kernel]
umem pid [textfile]

DESCRIPTION
Leak examines the named processes, which should be sharing their data and bss segments, for
memory leaks. It uses a mark and sweep-style algorithm to determine which allocated blocks are
no longer reachable from the set of root pointers. The set of root pointers is created by looking
through the shared bss segment as well as each process’s registers.

Unless directed otherwise, leak prints, for each block, a line with seven space-separated fields: the
string block, the address of the block, the size of the block, the first two words of the block, and
the function names represented by the first two words of the block. Usually, the first two words of
the block contain the malloc and realloc tags (see malloc(2)), useful for finding who allocated the
leaked blocks.

If the -s or the -c option is given, leak will instead present a sequence of acid(1) commands that
show each leaky allocation site. With -s a comment appears next to each command to indicate
how many lost blocks were allocated at that point in the program. With -c the comments are
extended to indicate also the total number of bytes lost at that point in the program, and an addi-
tional comment line gives the overall total number of bytes.

If the -a option is given, leak will print information as described above, but for all allocated blocks,
not only leaked ones. If the -d option is given, leak will print information as described above, but
for all free blocks, i.e. those freed, or those that are not yet in use (fragmentation?). The -a and
-d options can be combined.

If the -b option is given, leak will print a Plan 9 image file graphically summarizing the memory
arenas. In the image, each pixel represents res (default 8) bytes. The color code is:
dark blue: Completely allocated.
bright blue: Contains malloc headers.
bright red: Contains malloc headers for leaked memory.
dark red: Contains leaked memory.
yellow: Completely free.
white: Padding to fill out the image. The bright pixels representing headers help in counting
the number of blocks. Magnifying the images with lens(1) is often useful.

If given a name rather than a list of process ids, leak echoes back a command-line with process
ids of every process with that name.

The -f option specifies a binary to go on the acid(1) command-line used to inspect the processes,
and is only necessary when inspecting processes started from stripped binaries.

Umem prints a summary of all allocated blocks in the process with id pid. Each line of the sum-
mary gives the count and total size of blocks allocated at an allocation point. The list is sorted by
count in decreasing order. Umem prints summarizes all allocations, not just memory leaks, but it
is faster and requires less memory than leak.

Kmem is like umem but prints a summary for the running kernel.

EXAMPLES
List lost blocks in 8.out. This depends on the fact that there is only once instance of 8.out running;
if there were more, the output of leak -s 8.out would need editing before sending to the shell.

% leak -s 8.out
leak -s 229 230
% leak -s 8.out | rc
src(0x0000bf1b); // 64
src(0x000016f5); // 7
src(0x0000a988); // 7
%
View the memory usage graphic for the window system.
   % leak -b rio | rc | page
List the top allocation points in the kernel, first by count and then by total size:
   % kmem | sed 10q
   % kmem | sort -nr +1 | sed 10q

SOURCE
/sys/lib/acid/leak
/sys/src/cmd/aux/acidleak.c
/rc/bin/leak
/rc/bin/kmem
/rc/bin/umem

SEE ALSO
getcallerpc(2), setmalloctag in malloc(2)

BUGS
Leak and kmem depend on the internal structure of the libc pool memory allocator (see pool(2)).
Since the ANSI/POSIX environment uses a different allocator, leak will not work on APE programs.
Leak is not speedy, and acidleak can consume more memory than the process(es) being examined.
These commands require /sys/src/libc/port/pool.acid to be present and generated from pool.c.
NAME
  lens – interactive screen magnifier

SYNOPSIS
  lens

DESCRIPTION
  *Lens* presents a magnified view in its window of an arbitrary area on the screen. The default magnification is 4 (showing each pixel as a 4×4 pixel block in *lens*'s window). This may be changed by typing a digit on the keyboard (with 0 standing for 10), or by using the + and – keys to increase or decrease the magnification by one unit. The lower limit is ×1; the upper ×16.

  The interface to indicate what area to magnify is dictated by the mouse multiplexing rules of *rio*(1). Start by pressing mouse button 1 in the *lens* window and dragging, with the button pressed, to the center of the area to magnify. *Lens* will update the display as the mouse moves. Releasing the button freezes the *lens* display. The magnified view is static—a snapshot, not a movie—but typing a space or . key in the *lens* window will refresh the display, as will changing the magnification.

  To make counting pixels easier, typing a g toggles whether a checkerboard grid is imposed on the magnified area.

  Button 3 brings up a menu of actions.

SOURCE
  /sys/src/cmd/lens.c

BUGS
  There should be an easier way to indicate what to magnify.
NAME
lex – generator of lexical analysis programs

SYNOPSIS
lex [−tvn9] [file ...]

DESCRIPTION
Lex generates programs to be used in simple lexical analysis of text. The input files (standard input default) contain regular expressions to be searched for and actions written in C to be executed when expressions are found.

A C source program, lex.yy.c is generated. This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

The options have the following meanings.
−t Place the result on the standard output instead of in file lex.yy.c.
−v Print a one–line summary of statistics of the generated analyzer.
−n Opposite of −v; −n is default.
−9 Adds code to be able to compile through the native C compilers.

EXAMPLES
This program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```c
%%
[A−Z]   putchar(yytext[0] + 'a'−'A');
[ ]+$
[ ]+  putchar(' ');
```

FILES
lex.yy.c     output
/sys/lib/lex/ncform   template

SEE ALSO
yacc(1), sed(1)

SOURCE
/sys/src/cmd/lex

BUGS
Cannot handle UTF.
The asteroid to kill this dinosaur is still in orbit.
NAME
  lock – run a command under lock

SYNOPSIS
  lock [−w] lockfile [ command [ argument ... ]] 

DESCRIPTION
  Lock runs command (default rc) with arguments while holding lockfile open and (over)writing at least one byte each minute to keep the exclusive-access lock alive. If lockfile doesn’t already have the exclusive-access bit set in its mode, the exclusive-access bits are set in its mode and qid.type.

  Under −w, lock waits for exclusive access to lockfile instead of just trying once.

  Lock sets /env/prompt to contain the name of the lock file.

EXAMPLES
  Build a replica(1) database while preventing collisions with other occurrences.
    cd /sys/lib/dist
    lock scan.lock replica/scan $dist/sources.replica

SOURCE
  /sys/src/cmd/lock.c

SEE ALSO
  intro(5), stat(5)
NAME
look - find lines in a sorted list

SYNOPSIS
look [-dfnixtc][string][file]

DESCRIPTION
Look consults a sorted file and prints all lines that begin with string. It uses binary search.
The following options are recognized. Options dfnt affect comparisons as in sort(1).
- i Interactive. There is no string argument; instead look takes lines from the standard input
as strings to be looked up.
- x Exact. Print only lines of the file whose key matches string exactly.
- d ‘Directory’ order: only letters, digits, tabs and blanks participate in comparisons.
- f Fold. Upper case letters compare equal to lower case.
- n Numeric comparison with initial string of digits, optional minus sign, and optional decimal
point.
- t[c] Character c terminates the sort key in the file. By default, tab terminates the key. If c is
missing the entire line comprises the key.

If no file is specified, /lib/words is assumed, with collating sequence df.

FILES
/lib/words

SOURCE
/sys/src/cmd/look.c

SEE ALSO
sort(1), grep(1)

DIAGNOSTICS
The exit status is “not found” if no match is found, and “no dictionary” if file or the
default dictionary cannot be opened.
NAME
lp - printer output

SYNOPSIS
lp [ option ... ] [ file ... ]

DESCRIPTION
Lp is a generalized output printing service. It can be used to queue files for printing, check a queue, or kill jobs in a queue. The options are:

-\ d  dest Select the destination printer. If dest is ?, list the currently available printers. In the absence of -d, the destination is taken from the environment variable LPDEST. Destination stdout is the standard output. Destination safari is /dev/lpt1data line printer port on a 386 machine, assumed to be connected to a PostScript printer. Destinations hpdeskjet and bjc240l are also /dev/lpt1data but assumed to be connected to an HP Deskjet 670 or Canon BJC-240. Lp can print to any printer supported by Ghostscript using syntax gs!device where device is a Ghostscript output device. See gs(l) and the canonnbc240l entry in /sys/src/cmd/lp.

-\ k Kill the job(s) given as subsequent arguments, instead of file names, for the given destination.

-\ p  proc The given processor is invoked. The default processor is generic, which tries to do the right thing for regular text, troff() output, or tex(1) output. If no processing is desired npproc may be specified.

-\ q Print the queue for the given destination. For some devices, include printer status.

-\ R Stops and restarts the printer daemon. If the printer is wedged, it is often useful to cycle the power on the printer before running this command.

The remaining options may be used to affect the output at a given device. These options may not be applicable to all devices.

-\ c  n  Print n copies.

-\ f  font Set the font (default CW .11).

-\ H Suppress printing of header page.

-\ i  n  Select paper input tray. n may be a number 0–9, the word man for the manual feed slot, and/or simplex or duplex to get single or double sided output. Multiple input tray options may be specified if they are separated by commas.

-\ l  n  Set the number of lines per page to n.

-\ L Print pages in landscape mode (i.e. turned 90 degrees).

-\ m  v  Set magnification to v.

-\ n  n  Print n logical pages per physical page.

-\ o  list Print only pages whose page numbers appear in the comma-separated list of numbers and ranges. A range n–m means pages n through m; a range –n means from the beginning to page n; a range n– means from page n to the end.

-\ r Reverse the order of page printing.

-\ x  v  Set the horizontal offset of the print image, measured in inches.

-\ y  v  Set the vertical offset of the print image, measured in inches.

EXAMPLES
eqn paper | troff \-ms | lp \-dsafari
Typeset and print a paper containing equations.
pr \-1100 file | lp \-1100 \-fC8 .8
Print a file in a small font at 100 lines per page.
lp \-dstdout /dev/windows/3/window > doc.ps
Convert an image to a postscript file.

SOURCE
\rc\bin\lp
\sys\src\cmd\lp

SEE ALSO
lp(8)
P. Glick, “A Guide to the Lp Printer Spooler”.
BUGS
Not all options work with all output devices. Any user can kill any job.
NAME
ls, lc - list contents of directory

SYNOPSIS
ls [-dlmnpqrstuvwxyzFQT] name ...
lc [-dlmnpqrstuvwxyzFQT] name ...

DESCRIPTION
For each directory argument, ls lists the contents of the directory; for each file argument, ls repeats its name and any other information requested. When no argument is given, the current directory is listed. By default, the output is sorted alphabetically by name.

Lc is the same as ls, but sets the –p option and pipes the output through mc(1).

There are a number of options:

- If argument is a directory, list it, not its contents.
- List in long format, giving mode (see below), file system type (e.g., for devices, the # code letter that names it; see intro(3)), the instance or subdevice number, owner, group, size in bytes, and time of last modification for each file.
- List the name of the user who most recently modified the file.
- Don’t sort the listing.
- Print only the final path element of each file name.
- List the qid (see stat(2)) of each file; the printed fields are in the order path, version, and type.
- Reverse the order of sort.
- Give size in Kbytes for each entry.
- Sort by time modified (latest first) instead of by name.
- Under –t sort by time of last access; under –l print time of last access.
- Add the character / after all directory names and the character * after all executable files.
- Print the character t before each file if it has the temporary flag set, and – otherwise.
- By default, printed file names are quoted if they contain characters special to rc(1). The –Q flag disables this behavior.

The mode printed under the –l option contains 11 characters, interpreted as follows: the first character is

d if the entry is a directory;
a if the entry is an append-only file;
- if the entry is a plain file.

The next letter is 1 if the file is exclusive access (one writer or reader at a time).

The last 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user–group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, ‘execute’ permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
- if none of the above permissions is granted.

SOURCE
/sys/src/cmd/ls.c
/rc/bin/lc
SEE ALSO

\texttt{stat(2)}, \texttt{mc(1)}
NAME
mail, go.fishing - mail and mailboxes

SYNOPSIS
mail [ arg ...]
go.fishing

DESCRIPTION
Mail is a shell script that invokes nedmail(1), the mail reader, when no recipients appear on the
command line and marshal(1), the mail preparer, otherwise. All command line options are passed
through. See the man pages for those two commands for more details.

Incoming mail for a user username is put in the file /mail/box/username/mbox unless either the
file /mail/box/username/forward or /mail/box/username/pipeto exists. The
mailbox must have append-only and exclusive-access mode (see chmod(1)). A user must create
his or her own mailbox using the -c option of nedmail(1). Mailboxes are created writable
(append-only) but not readable by others.

If the file /mail/box/username/forward exists and is readable by everyone, incoming mail
will be forwarded to the addresses contained in the first line of the file. The file may contain multiple
addresses. Forwarding loops are caught and resolved by local delivery.

If the file /mail/box/username/pipeto exists and is readable and executable by everyone, it
will be run for each incoming message for the user. The message will be piped to it rather than
 appended to his/her mail box. The file is run as user none. Its two arguments are the with arguments of the destination address (e.g., local!gremlin) and the user's mail box path (e.g.,
/mail/box/gremlin/mbox)

Auto-answer
To use mail as an answering machine while you are away, run go.fishing, which will create
/mail/box/$user/gone.fishing as a flag for pipeto processing, and truncate
/mail/box/$user/gone.addrs. Any existing pipeto file that uses
/mail/lib/pipeto.lib will invoke the gone.fishing machinery when it calls spool or spool-tagged-spam.

If /mail/box/$user/gone.msg exists, it will be sent (just once) to everyone who sends you
mail that lists your address in a To or Cc header; if not, /mail/lib/gone.msg will be sent.
Upon your return, remove /mail/box/$user/gone.fishing to stop automatic responses.

FILES
/sys/log/mail mail log file
/mail/box/* mail directories
/mail/box/*/mbox mailbox files
/mail/box/*/forward forwarding address(es)
/mail/box/*/pipeto mail filter
/mail/box/*/L.reading mutual exclusion lock for multiple mbox readers
/mail/box/*/L.mbox mutual exclusion lock for altering mbox
/lib/face/48x48x? directories of icons for screen
/mail/lib/pipeto.lib helper functions for pipeto files
/mail/lib/gone.msg default vacation message
/mail/lib/gone.fishing auto-responder as pipeto script
/mail/box/$user/gone.fishing flag to active gone processing
/mail/box/$user/gone.addrs list of senders answered by gone.fishing

SOURCE
/rc/bin/mail
/rc/bin/go.fishing

SEE ALSO
aliasmail(8), faces(1), filter(1), marshal(1), mlmgr(1), nedmail(1), qer(8), rewrite(6), send(8), smtp(8), upasfs(4)
NAME
mailcmd – mail the output of a failed command

SYNOPSIS
mailcmd [-m addr]... command line

DESCRIPTION
Mailcmd runs the command line using rc(1). If the command exits with an error status, mailcmd
mails the command’s output to the named addresses (by default, the current user).

Mailcmd is typically used to run automatic scripts such as those started by cron(8).

SOURCE
/rc/bin/mailcmd

SEE ALSO
cron(8)
NAME
man, lookman, sig - print or find pages of this manual

SYNOPSIS
man [ −bnpPStw] [ section ... ] title ...
lookman key ...
sig function ...

DESCRIPTION
Man locates and prints pages of this manual named title in the specified sections. Title is given in lower case. Each section is a number; pages marked (2S), for example, belong to chapter 2. If no section is specified, pages in all sections are printed. Any name from the NAME section at the top of the page will serve as a title.

The options are:
−n (Default) Print the pages on the standard output using nroff.
−b Print the pages using nroff and send them to plumber(4) for display in the editor.
−p Run proof(1) on the specified man pages.
−P Run page(1) on the specified man pages.
−S Do not search the manual indices for the names. Only print pages whose file names match the names.
−t Run troff(1) and send its output to standard output.
−w Print the names of the man page source files.

Lookman prints the names of all manual sections that contain all of the key words given on the command line.

Sig prints the signature (i.e. C definition) of the functions given on the command line.

FILES
/sys/man/?/*
  troff source for manual; this page is /sys/man/1/man
/sys/man/?/INDEX
  indices searched to find pages corresponding to titles
/sys/lib/man/secindex
  command to make an index for a given section
/sys/lib/man/lookman/index
  index for lookman

SOURCE
/rc/bin/man
/rc/bin/lookman

SEE ALSO
page(1), proof(1)

BUGS
The manual was intended to be typeset; some detail is sacrificed on text terminals.
There is no automatic mechanism to keep the indices up to date.
Except for special cases, man doesn’t recognize things that should be run through tbl and/or eqn.
NAME
marshal - formatting and sending mail

SYNOPSIS
upas/marshal[[-aA] attachment][[-C copyaddr][[-nrx#][-R reply-msg][-s subject]
[-t mime-type][ mailaddr ...]]

DESCRIPTION
Marshal builds a mail message from standard input and passes it, if the body is non-empty, for
transmission or delivery to /mail/box/username/pipefrom if it exists, otherwise to
/bin/upas/send. The message format is both RFC 822 and MIME conformant, so marshal
adds any required headers not already in the message, prefixed by the contents of
/mail/box/username/headers. This allows the addition of personal headers like From:
lines with a full name or a different return address. Command line options direct marshal to add a
subject line and append attachments. The arguments to marshal are the addresses of the recipients.

When running in a rio(1) window, marshal automatically puts the window into hold mode (see
rio(1)); this means that the message can be edited freely, because nothing will be sent to marshal
until the ESC key is hit to exit hold mode.

The options are:
-af
   directs marshal to append file as a mime attachment. Unless explicitly specified by
   the -t option, the type of the attachment is determined by running the file(1) com-
   mand.
-Afile
   is like -a but the message disposition is marked as inline directing any mail reader
to display the message (if it can) when the mail message is read.
-Ccopyaddr
   adds a Cc: header with copyaddr and also adds copyaddr as a recipient.
-n
   intentionally no standard input
-xr
   are all passed as command line options to the send that marshal invokes.
-Rreplymsg
   tells marshal what message this one is in reply to. Replymsg is an upasfs(4) directory
containing the message. Marshal uses any message id in this message in its
In-Reply-To field. It also passes the directory to
/mail/box/username/pipefrom in the replymsg environment variable. Thus, pipefrom can alter the message to somehow match the reply to the message
it is replying to.
-ssubject
   adds a Subject: header line to the message if one does not already exist.
-ttype
   sets the content type for the attachments from all subsequent -a and -A options.

Marshal also expands any user mail aliases contained in /mail/box/username/names. The
format of the alias file is the same as that for system aliases, see aliasmail(8).

Marshal uses the login name as the reply address. This can be overridden using the environment
variable upasname. Its value will become both the envelope and From: mailbox name. For
example:

    upasname=natasha@kremvax.com upas/mail boris@squirrel.com

FILES
/mail/box/*/dead.letter

SOURCE
/sys/src/cmd/upas/marshal

SEE ALSO
    aliasmail(8), faces(1), filter(1), mail(1), mlmgr(1), nedmail(1), qer(8), rewrite(6), send(8), smtp(8),
    upasfs(4)
NAME
   mc – multicolumn print

SYNOPSIS
   mc [−] [−N] [ file ...]

DESCRIPTION
   Mc splits the input into as many columns as will fit in N print positions. If run in a rio(1) or
   acme(1) window, the default N is the number of blanks that will fit across the window; otherwise
   the default N is 80. Under option – each input line ending in a colon : is printed separately.

SOURCE
   /sys/src/cmd/mc.c

SEE ALSO
   rio(1), acme(1), acme(4), pr(1), lc in ls(1)
NAME
mk, membername - maintain (make) related files

SYNOPSIS
mk [-f mkfile] ... [ option ... ] [ target ... ]
membername aggregate ...

DESCRIPTION
Mk uses the dependency rules specified in mkfile to control the update (usually by compilation) of
targets (usually files) from the source files upon which they depend. The mkfile (default mkfile)
contains a rule for each target that identifies the files and other targets upon which it depends and
an rc(1) script, a recipe, to update the target. The script is run if the target does not exist or if it
is older than any of the files it depends on. Mkfile may also contain meta-rules that define actions
for updating implicit targets. If no target is specified, the target of the first rule (not meta-rule) in
mkfile is updated.

The environment variable $NPROC determines how many targets may be updated simultaneously;
Plan 9 sets $NPROC automatically to the number of CPUs on the current machine.

Options are:
-a Assume all targets to be out of date. Thus, everything is updated.
-d[egp] Produce debugging output (p is for parsing, g for graph building, e for execution).
-e Explain why each target is made.
-i Force any missing intermediate targets to be made.
-k Do as much work as possible in the face of errors.
-n Print, but do not execute, the commands needed to update the targets.
-s Make the command line arguments sequentially rather than in parallel.
-t Touch (update the modified date of) file targets, without executing any recipes.
-wtarget1, target2,... Pretend the modify time for each target is the current time; useful in conjunction with
-n to learn what updates would be triggered by modifying the targets.

The rc(1) script membername extracts member names (see ‘Aggregates’ below) from its argu-
ments.

The mkfile
A mkfile consists of assignments (described under ‘Environment’) and rules. A rule contains
targets and a tail. A target is a literal string and is normally a file name. The tail contains zero or
more prerequisites and an optional recipe, which is an rc script. Each line of the recipe must
begin with white space. A rule takes the form

    target: prereq1 prereq2
    rc recipe using prereq1, prereq2 to build target

When the recipe is executed, the first character on every line is elided.

After the colon on the target line, a rule may specify attributes, described below.

A meta-rule has a target of the form A%B where A and B are (possibly empty) strings. A meta-rule
acts as a rule for any potential target whose name matches A%B with % replaced by an arbitrary
string, called the stem. In interpreting a meta-rule, the stem is substituted for all occurrences of %
in the prerequisite names. In the recipe of a meta-rule, the environment variable $stem contains
the string matched by the %. For example, a meta-rule to compile a C program using 2c(1) might be:

    %: %.c
    2c $stem.c
    2l -o $$stem $stem.2

Meta-rules may contain an ampersand & rather than a percent sign %. A % matches a maximal
length string of any characters; an & matches a maximal length string of any characters except
period or slash.

The text of the mkfile is processed as follows. Lines beginning with < followed by a file name are
replaced by the contents of the named file. Lines beginning with < followed by a file name are
replaced by the output of the execution of the named file. Blank lines and comments, which run from unquoted # characters to the following newline, are deleted. The character sequence backslash-newline is deleted, so long lines in `mkfile` may be folded. Non-recipe lines are processed by substituting for `{ command }` the output of the `command` when run by `rc`. References to variables are replaced by the variables’ values. Special characters may be quoted using single quotes ‘ ’ as in `rc()`. Assignments and rules are distinguished by the first unquoted occurrence of : (rule) or = (assignment).

A later rule may modify or override an existing rule under the following conditions:

- If the targets of the rules exactly match and one rule contains only a prerequisite clause and no recipe, the clause is added to the prerequisites of the other rule. If either or both targets are virtual, the recipe is always executed.
- If the targets of the rules match exactly and the prerequisites do not match and both rules contain recipes, `mk` reports an “ambiguous recipe” error.
- If the target and prerequisites of both rules match exactly, the second rule overrides the first.

Environment

Rules may make use of `rc` environment variables. A legal reference of the form `$OBJ` is expanded as in `rc()`. A reference of the form `$name:A%B=C%D`, where `A`, `B`, `C`, `D` are (possibly empty) strings, has the value formed by expanding `$name` and substituting `C` for `A` and `D` for `B` in each word in `$name` that matches pattern `A%B`.

Variables can be set by assignments of the form

```
var=[attr]=value
```

Blanks in the `value` break it into words, as in `rc` but without the surrounding parentheses. Such variables are exported to the environment of recipes as they are executed, unless `U`, the only legal attribute `attr`, is present. The initial value of a variable is taken from (in increasing order of precedence) the default values below, `mk`’s environment, the `mkfiles`, and any command line assignment as an argument to `mk`. A variable assignment argument overrides the first (but not any subsequent) assignment to that variable.

The variable `MKFLAGS` contains all the option arguments (arguments starting with – or containing =) and `MKARGS` contains all the targets in the call to `mk`.

It is recommended that `mkfiles` start with

```
</$objc/mkfile
```

to set CC, LD, AS, O, YACC, and MK to values appropriate to the target architecture (see the examples below).

Execution

During execution, `mk` determines which targets must be updated, and in what order, to build the `names` specified on the command line. It then runs the associated recipes.

A target is considered up to date if it has no prerequisites or if all its prerequisites are up to date and it is newer than all its prerequisites. Once the recipe for a target has executed, the target is considered up to date.

The date stamp used to determine if a target is up to date is computed differently for different types of targets. If a target is virtual (the target of a rule with the V attribute), its date stamp is initially zero; when the target is updated the date stamp is set to the most recent date stamp of its prerequisites. Otherwise, if a target does not exist as a file, its date stamp is set to the most recent date stamp of its prerequisites, or zero if it has no prerequisites. Otherwise, the target is the name of a file and the target’s date stamp is always that file’s modification date. The date stamp is computed when the target is needed in the execution of a rule; it is not a static value.

Nonexistent targets that have prerequisites and are themselves prerequisites are treated specially. Such a target `t` is given the date stamp of its most recent prerequisite and if this causes all the targets which have `t` as a prerequisite to be up to date, `t` is considered up to date. Otherwise, `t` is made in the normal fashion. The -i flag overrides this special treatment.
Files may be made in any order that respects the preceding restrictions.
A recipe is executed by supplying the recipe as standard input to the command
/bin/rc -e -I
(the -e is omitted if the E attribute is set). The environment is augmented by the following variables:

$alltarget
   all the targets of this rule.

$newprereq
   the prerequisites that caused this rule to execute.

$newmember
   the prerequisites that are members of an aggregate that caused this rule to execute. When
   the prerequisites of a rule are members of an aggregate, $newprereq contains the name of the aggregate
   and out of date members, while $newmember contains only the name of the members.

$nproc
   the process slot for this recipe. It satisfies 0≤$nproc<$NPROC.

$pid
   the process id for the mk executing the recipe.

$prereq
   all the prerequisites for this rule.

$stem
   if this is a meta-rule, $stem is the string that matched % or &. Otherwise, it is empty. For regular
   expression meta-rules (see below), the variables stem0, ..., stem9 are set to the corresponding subexpressions.

$target
   the targets for this rule that need to be remade.

These variables are available only during the execution of a recipe, not while evaluating the mkfile.

Unless the rule has the Q attribute, the recipe is printed prior to execution with recognizable environment variables expanded. Commands returning nonempty status (see intro(1)) cause mk to terminate.

Recipes and backquoted rc commands in places such as assignments execute in a copy of mk's environment; changes they make to environment variables are not visible from mk.

Variable substitution in a rule is done when the rule is read; variable substitution in the recipe is done when the recipe is executed. For example:

\begin{verbatim}
bar=a.c
foo: $bar
   $CC -o foo $bar
bar=b.c
\end{verbatim}

will compile b.c into foo, if a.c is newer than foo.

Aggregates
Names of the form a(b) refer to member b of the aggregate a. Currently, the only aggregates supported are ar(1) archives.

Attributes
The colon separating the target from the prerequisites may be immediately followed by attributes
and another colon. The attributes are:

D If the recipe exits with a non–null status, the target is deleted.
E Continue execution if the recipe draws errors.
N If there is no recipe, the target has its time updated.
n The rule is a meta–rule that cannot be a target of a virtual rule. Only files match the pattern in the target.

P The characters after the P until the terminating : are taken as a program name. It will be
invoked as rc -c prog 'arg1' 'arg2' and should return a null exit status if and only if arg1 is up to date with respect to arg2. Date stamps are still propagated in the normal way.
The recipe is not printed prior to execution.

The rule is a meta-rule using regular expressions. In the rule, % has no special meaning. The target is interpreted as a regular expression as defined in `regexp(6)`. The prerequisites may contain references to subexpressions in form \\n, as in the substitute command of `sam(1)`.

The targets are considered to have been updated even if the recipe did not do so.

The targets of this rule are marked as virtual. They are distinct from files of the same name.

**EXAMPLES**

A simple `mkfile` to compile a program:

```bash
&lt;/objtype/mkfile

    $LD $LDFLAGS -o $target $prereq

%.SO: %.c
    $CC $CFLAGS $stem.c

Override flag settings in the `mkfile`:

% mk target 'CFLAGS=-S -W'

Maintain a library:

```bash
libc.a(%.SO):N: %.SO
libc.a: libc.a(abs.$O) libc.a(access.$O) libc.a(alarm.$O) ...
    ar r libc.a $newmember
```

String expression variables to derive names from a master list:

```bash
NAMES=alloc arc bquote builtins expand main match mk var word
OBJ=${NAMES:%=%.$O}
```

Regular expression meta-rules:

```bash
([^/]*)/(.*)\.$O:R: \1/\2.c
cd $stem1; $CC $CFLAGS $stem2.c
```

A correct way to deal with `yacc(1)` grammars. The file `lex.c` includes the file `x.tab.h` rather than `y.tab.h` in order to reflect changes in content, not just modification time.

```bash
lex.$O: x.tab.h
x.tab.h: y.tab.h
    cmp -s x.tab.h y.tab.h || cp y.tab.h x.tab.h
    y.tab.c y.tab.h: gram.y
    $YACC -d gram.y
```

The above example could also use the `P` attribute for the `x.tab.h` rule:

```bash
x.tab.h: Pcmp -s: y.tab.h
    cp y.tab.h x.tab.h
```

**SOURCE**

/sys/src/cmd/mk

**SEE ALSO**

`rc(1)`, `regexp(6)`

A. Hume, "Mk: a Successor to Make".

Andrew G. Hume and Bob Flandrena, "Maintaining Files on Plan 9 with Mk".

**BUGS**

Identical recipes for regular expression meta-rules only have one target.

Seemingly appropriate input like `CFLAGS=-DHZ=60` is parsed as an erroneous attribute; correct it by inserting a space after the first `=`.

134
The recipes printed by \texttt{mk} before being passed to \texttt{rc} for execution are sometimes erroneously expanded for printing. Don't trust what's printed; rely on what \texttt{rc} does.
NAME
  mkdir – make a directory

SYNOPSIS
  mkdir [−p][−m mode] dirame ...

DESCRIPTION
  Mkdir creates the specified directories. It requires write permission in the parent directory.

  If the −p flag is given, mkdir creates any necessary parent directories and does not complain if the
target directory already exists.

  The −m flag sets the permissions to be used when creating the directory. The default is 0777.

SEE ALSO
  rm(1)
  cd in rc(1)

SOURCE
  /sys/src/cmd/mkdir.c

DIAGNOSTICS
  Mkdir returns null exit status if all directories were successfully made. Otherwise it prints a diag-
nostic and returns "error" status.
NAME
ml, mlmgr, mlowner - unmoderated mailing lists

SYNOPSIS
upas/mlmgr -c listname
upas/mlmgr -ar listname address
upas/ml [-r replyto-address] addressfile listname
upas/mlowner addressfile listname

DESCRIPTION
Mlmgr creates and updates unmoderated mailing lists. The -c option creates mail directories for both listname and listname-owner, each containing a pipeto file. Messages mailed to listname are sent to all members of the mailing list. Any Reply-to: and Precedence: fields are removed from the messages and new ones are added directing replies to listname and specifying bulk precedence. The envelope address for error replies is set to /dev/null.

The mailing list membership is the file /mail/box/listname/address-list. This file is an add/remove log. Each line represents a single address. Lines beginning with a hash (#) are comments. Lines beginning with an exclamation point (!) are removals. All other lines are additions.

Addition and removal entries can be appended using the -a and -r options to mlmgr. However, they are normally appended as a consequence of user requests.

To be added or removed from the list, a user may send a message to listname-owner containing a key word in the header or body. The key words are:

subscribe - add my From: address to the list
remove - remove my From: address from the list
unsubscribe - remove my From: address from the list

Addition and removal events cause notification messages to be sent to the added/removed address. In the case of addition, the message describes how to be removed.

Ml and mlowner are the programs that receive messages for listname and listname-owner respectively. Appropriate calls to them are inserted in the pipeto files created by mlmgr.

Ml’s -r option sets the Reply-to: field in the mail sent out by ml.

FILES
/mail/box/<listname> list’s mailbox directory
/mail/box/<listname>-owner owner’s mailbox directory
/mail/box/<listname>/address-list log of mailing list deletions and additions

SOURCE
/sys/src/cmd/upas/ml

SEE ALSO
aliasmail(8), faces(1), filter(1), mail(1), marshal(1), nedmail(1), qer(8), rewrite(6), send(8), smtp(8), upasfs(4)
NAME
mp3dec - decode audio MPEG files (layers 1, 2 and 3)

SYNOPSIS
mp3dec [ -o outfile ] [ file ... ]

DESCRIPTION
Mp3dec decodes one or more MPEG audio files, writing 16-bit stereo linear PCM sample data to outfile (default /dev/audio). If no files are named, mp3dec reads standard input.

In the absence of the -o option, mp3dec also opens /dev/volume and sets the sample rate for playback to match the audio stream. When writing to outfile, mp3dec prints a warning to standard error if the stream rate is not 44100 Hz.

SOURCE
/sys/src/games/mp3dec

SEE ALSO
mp3enc(1), juke(7), playlists(7)
http://www.underbit.com/products/mad/

BUGS
It's another GNU behemoth, lightly tamed.
NAME
mp3enc - create mp3 audio files

SYNOPSIS
 games/mp3enc [-hpvr] [-b bitrate] [-B bitrate] [-m mode] [-q] [-s sfreq] [-V q] [long or silly options]

DESCRIPTION
Mp3enc compresses audio on standard input, normally PCM-encoded, and produces MP3-encoded audio on standard output. By default, the MP3 file will use 'constant bit-rate' (CBR) encoding, but that can be changed via --abr (average bitrate desired, ABR) or --v (variable bitrate, VBR).

Options

- b  set minimum allowed bitrate in Kb/s for VBR, default 32Kb/s. For CBR, set the exact bitrate in Kb/s, which defaults to 128Kb/s.
- B  set maximum allowed bitrate in Kb/s for VBR, default 256Kb/s.
- h  same as --q 2.
- m  mode may be (s)tereo, (j)oint, (f)orce or (m)ono (default j). force forces mid/side stereo on all frames.
- p  add CRC error protection (adds an additional 16 bits per frame to the stream). This seems to break playback.
- q  sets output quality to q (see --V).
- r  input is raw pcm
- s  set sampling frequency of input file (in KHz) to sfreq, default is 44.1.
- v  use variable bitrate (VBR) encoding
- V  set quality setting for VBR to q. Default q is 4; 0 produces highest-quality and largest files, and 9 produces lowest-quality and smallest files.

Long options

--abr bitrate  sets average bitrate desired in Kb/s, instead of setting quality, and generates ABR encoding.
--resample sfreq  set sampling frequency of output file (in KHz) to sfreq, default is input sfreq.
--mp3input  input is an MP3 file

Silly options

- f  same as --q 7. Such a deal.
- o  mark as non--original (i.e. do not set the original bit)
- c  mark as copyright
- k  disable sfb=21 cutoff
- e emp  de-emphasis n/5/c (default n)
- d  allow channels to have different blocktypes
- t  disable Xing VBR informational tag
- a  autoconvert from stereo to mono file for mono encoding
- x  force byte--swapping of input (see dd(1) instead)
- S  don't print progress report, VBR histograms
--athonly  only use the ATH for masking
--nohist  disable VBR histogram display
--voice  experimental voice mode

EXAMPLES
Encode a .wav file as highest-quality MP3.
 games/mp3enc -q 0 -b 320
Create a fixed 128Kb/s MP3 file from a .wav file.
 games/mp3enc -h <foo.wav >foo.mp3
Streaming from stereo 44.1KHz raw PCM data, encoding mono at 16KHz (you may not need dd):
 dd -conv swab | games/mp3enc -a -r -m m --resample 16 -b 24

SOURCE
/sys/src/games/mp3enc
SEE ALSO
   dd(1), mp3dec(1), audio(3), cdfs(4), audio(7), juke(7), playlistfs(7)
   http://www.sulaco.org/mp3

BUGS
   Quality is much better than encoders based on the ISO routines, but still not as good as the FhG encoder.
   It's a GNU behemoth, lightly rehabilitated. There are zillions of undocumented options.
NAME
ms2html, html2ms - convert between troff's ms macros and html

SYNOPSIS
ms2html [-q] [-b basename] [-d delims] [-t title]
html2ms

DESCRIPTION
Ms2html converts the ms(6) source on standard input into HTML and prints it to standard output. If
the source contains tbl(1) or eqn input, you must first pipe the text through those preprocessors. Postscript
images, equations, and tables will be converted to gif files. If the document has a .TL entry, its contents will
be used as the title; otherwise ms2html will look for a .T macro, unknown to ms(6), and take its value. Options are:
q  suppresses warnings about malformed input;
b  sets the HTML base name to basename;
d  sets the eqn(1) delimiters to delim;
t  sets the HTML title to title.

Html2ms reads HTML from standard input and converts it to ms(6) source on standard output.

SOURCE
/sys/src/cmd/ms2html.c
/sys/src/cmd/html2ms.c

SEE ALSO
htmlroff(1), ms(6)

BUGS
Ms2html doesn't understand a number of troff commands. It does handle macros and defined strings.
Html2ms doesn't understand html tables.
NAME

Msgs, mail2fs, M, Mg, mspool, mailplumb, msgs, Arch, Spam, Reply, Send, Post, Delmesg, Save – file based mail reader

SYNOPSIS

mail2fs [-acDnor][ -d mdir][ mbox]
msgs [-aD][ -s runes][ mdir][ monthdir]
M cmd [ dir . . .]
Mg [-h][ regexp]
mspool
mailplumb [-dho][ mdir]
Msgs [-a][ mdir[ monthdir]]
Arch [ file . . .]
Spam [ file . . .]
Reply
Send
Delmesg [ file . . .]
Post
Save file
mlist

DESCRIPTION

These programs cooperate to provide mail reading and delivering facilities by using files from a shared file server. Msgs are stored in a convenient way to read or process them just by browsing files, using a Plan B mail box format.

Mail box format

In Plan B, msgs for users are parsed and decoded first, and then stored in a file hierarchy where these and other tools can be used to process them. A mailbox is a directory, usually under /mail/box/$user/, that contains one directory per month (e.g., 200603/ for msgs processed on March 2006). In these directories there is one directory per message. The convention is that (message directory) names starting with a . correspond to archived messages not to be usually shown to the user. Names starting with s . correspond to messages that seem to be spam (not usually shown either). Names starting with d. correspond to deleted messages not yet removed from the file system. Any other rune can be used instead of a . , s or d as a convenience (the meaning would be up to the user). But for this optional prefix, messages use a serial message number as their directory name.

The directory for a message contains at least two files: text and raw. The text file has the mail headers and body already processed for reading, and raw has the original mail headers without any processing, including the UNIX header line (for debugging and also for obtaining message ids when replying to msgs). As an option, raw may contain the entire original mail without processing and not just the headers. Any attachment in the mail is kept stored in a separate file (possibly with the file name indicated in the MIME headers) ready to be used, that is, decoded. When the attachment is a mail, the message is stored in a subdirectory following the same conventions stated above. For msgs with attachments, the text file contains additional text indicating the relative path names (from the mail’s directory) that can be used to open the attachments. This is convenient to plumb(1) them while reading.

Because all these files have been already processed for reading, the usual file handling tools can be used to read, edit, copy, or remove them.

The mailbox used by default is /mail/box/$user/msg, and corresponds to the inbox.

A Plan B mail box also contains two files: seq and digest. Messages are given sequence numbers as added to the mail box. The file seq contains the sequence number for the last message (or zero) and is DMEXCL to provide locking for multiple programs using the mail box. The file
digest contains digests for msgs added to the mailbox using `mail2fs` (and not for those added by hand using file tools). When a message has a digest that was already seen in the past the message is silently discarded as a duplicate.

Programs described below are parsimonious enough in the format of the mailbox so that they will work even if messages are edited by hand, other files are created, or some of them are removed.

Virtual mail folders may be created by storing text files with mail lists that contain a mail description per line starting with the path for each mail. Copying the text shown for some messages in a mail listing into another text file would “save” such messages into that file. The program `mlist` writes to standard output a clean listing for messages with paths found in the standard input.

A file (a virtual folder) named like a per–month directory with `.1` or `.1a` appended is considered a cache of the listing for that directory. But only if the directory has not been modified after the file. For example, a text file named `200909.1` (containing a mail listing in the format shown by `msgs`) will be used to list `msgs` on September 2009, instead of the `200909` directory and its contents. This file will be ignored for mailbox listings if the directory is modified after the file and the cache is, therefore, out of date. A file named `200909.1a` behaves like before, but is known to contain only archived `msgs`. It is, therefore, ignored when listing non–archived `msgs`. The `msgs` program automatically renames `.1` files to `.1a when the file lists only archived mail.

**Reading mail**

Mail is converted from a Plan 9 mailbox into a Plan B mailbox using `mail2fs`. This program may be run using `cron(8)` or directly from the `pipe to` file described in `mail(1)`. It uses `upas/fs`, described in `upasfs(4)`, to parse the Plan 9 mail box.

By default `/mail/box/`$USER`/mbox` is the source (Plan 9) mail box. Messages found on it are moved into (in Plan B mailbox format). The argument `mbox` specifies another mail source, to be used instead of the default. Option `−d` changes the destination to `mdir`.

Messages are deleted from the Plan 9 mailbox unless flag `−n` is given. The Plan B mailbox is created if it does not exist only if flag `−c` is given.

Flag `−a` makes `mail2fs` add new messages as archived to the Plan B mailbox. This is useful to add messages to a mailbox for further reference, not for listing when asking the mail index for the mailbox. For example, to archive outgoing mail in the default mail box.

Flag `−o` asks `mail2fs` to use the date from the mail to determine where to archive it, instead of using today’s. See the examples section.

Flag `−r` asks `mail2fs` to store the entire original message in the `raw` file, and not just message headers.

Flag `−D` (accepted by `mail2fs` and other programs) enables debug diagnostics.

The program `msgs` is a convenience tool for reading mail. It generates a mail index. Flag `−a` generates a list for all `msgs` in the mailbox, archived or not. This may take some time if the mailbox is big enough. Flag `−s` can be used to include in the list `msgs` starting with any rune in `rune`s, for example, `s` for spam. The mailbox is the standard `msgs` inbox unless a different one is supplied as an argument. As an option, both the mail box path and the name of a per–month directory can be indicated, to ask for a list of `msgs` for just one month. This is useful to generate per–month mail lists that can be used to quickly browse existing `msgs` without going through all `msgs` in the box. See the `Month` script supplied as an example.

As an aid for other programs, `msgs` places a list of the directories for the `msgs` listed at `/tmp/msgs`.$USER`, which can be useful for retrieve the paths for the `msgs` the user is working with.

Programs just described are the basic toolkit to handle `msgs`. Other programs described here are a convenience for the user. Users are encouraged to customize them and/or to write their own versions.

`M` is a script that applies the operation indicated by `cmd` to one or more `msgs`. It applies `cmd` to all `msgs` last listed by `msgs`, (as described by the paths in `/tmp/msgs`.$USER`), when no mail directories are given as arguments. Arguments selecting `msgs` only need to mention paths to the mail directories, but may refer to particular files within them, as a convenience, to permit using names from somewhere else without editing. `Cmd` may be any of the following:
arch  To archive the msgs as read.
spam  To archive the msgs as spam.
inbox To archive the msgs as unread.
rm    To print commands to remove the msgs.
print To print the text of the msgs.
list  To list the directories for the msgs.
mime  To list the attachments for the msgs.
reply To plumb a reply message to the editor.

The single letters a, s, i, d, p, l, m, and r can be used instead of the full cmd name (in the same order). Note that the letter is the initial for the command, but for deletion.

Mg is not strictly necessary, but is supplied as a convenience script to call grep(1) to locate msgs containing the expression given as an argument. Flag –h makes it search only in headers. Like the previous program, Mg only considers msgs listed in /tmp/mgs.$user.

Mailplumb is used to send plumb(1) messages to maintain faces(1) and other programs aware of msgs in the user’s Plan B mailbox, or in mdir when supplied. Flag –h makes the program notify existing msgs as new ones. Flag –o makes mailplumb post events for the Octopus, using ports(4) instead of plumber(4).

Reading mail in Acme
The program Msgs (see /acme/msgs) is an Acme interface for reading mail. Its arguments are the same of msgs. Executing Msgs within Acme displays a window with the default (or indicated) mailbox. It understands both the standard mailbox format (described above) and the virtual folder format (a text file, see above).

The program listens to plumber events for mail and tries to maintain the mail list up to date.

To read a mail simply click button-3 on the mail file name shown in the mail list. Archiving, marking a spam, replying, and deleting messages is achieved by executing the scripts Arch, Spam, Reply and Delmesg respectively. Most of them may be used either from a message window or to process standard input (usually at the mail box directory). Some of them also accept message paths as arguments. For example, to archive a series of msgs, select them in the mail list and execute | Arch in that window.

Msgs can be sent using Reply and Post as described later.

Save archives one or more messages, adding them to the named file.

Reading mail in O/live
In general, reading mail in olive(1) is similar to reading mail in Acme. The main difference is that there is no need for a mail listing command. That is, there is no Msgs program.

Executing !Msgs at /mail/box/$user/msgs produces an initial list of msgs. This list can be refreshed by executing ,<Msgs for the panel containing the mail list. To read a mail just click (button-3) on the mail path.

To select msgs according to text shown in the mail index use the Sam command language. For example, ,x/9fans/+-p produces a mail index for msgs coming from 9fans.

To archive a set of msgs send their index text as standard input to Arch. For example, ,>Arch archives all msgs listed in the panel. In the same way, Spam flags msgs as spam.

As in Acme, most scripts can be used for a panel showing a single message, for standard input, or for messages given as argument.

Sending mail
Msgpool is a program that takes text files from /mail/box/$user/out representing msgs to be sent, and sends them. It only operates on files whose names are numbers. To send a mail, the user creates a file with a randomized name like /mail/box/$user/out/Out.3452, edits it, and renames the file to just the random number.

The file format is similar to that used by the acme(1) mail composition window. It includes one text line per header, a blank line, and the body. Attachments are added by lines starting with Attach: in the header. Inline attachments are added by lines starting with Include: in the header. Replies to other msgs should contain a Replying: header containing the path to the mail being replied to (its raw file in a Plan B mailbox).
Messages are sent using `marshal(1)`.

The script `Reply` is available to send messages from either `olive(1)` or `acme(1)`. Similar to `Arch` and `Spam`, it replies to the mail shown when executed in a message window (or panel); it replies to the mail listed in its standard input otherwise. For example, selecting a mail in the index and executing `.>Reply` in `Olive` or `.>Reply` in `Acme` would reply to it.

When uncertain regarding the mail to reply to, it creates a window to edit and send a new mail.

Mail is delivered by writing the panel created by `Reply` and then executing either `Send` or `Post`. The former spools the message using `mspool`, the latter attempts to immediately deliver the message.

**EXAMPLES**

Users are expected to customize the scripts supplied to their needs. All of them are to be considered examples of how to use the mail system.

Move all msgs from the Plan 9 mailbox to the Plan B one, and creates the later if it does not exist.

```
; mail2fs -c
List msgs:
; msgs omsgs
```

```
200712/4/text        Ralph Corderoy       Re: [9fans] Hi
200712/3/text        Juan Manuel Se       Re: reunion
200712/2/text        "Raquel Martin        Re: [Diet] reunion
200712/1/text        "Fco. J. Balle        reunion
```

From now on, `/tmp/msgs.$user` contains a list of mail directories for `M` to work with. For example, display them.

```
; M p
/mail/box/nemo/msgs/200712/4
To: 9fans@cse.psu.edu
From: Ralph Corderoy <ralph@inputplus.co.uk>
Subject: Re: [9fans] Hi together | a few newbie questions
Sender: 9fans-admin@cse.psu.edu
...
List their directories and plumb all PDF attachments:
; M l
```

```
/mail/box/nemo/msgs/200712/4
/mail/box/nemo/msgs/200712/3
/mail/box/nemo/msgs/200712/2
/mail/box/nemo/msgs/200712/1
```

```
plumb '{M l}^*.pdf
```

Reply to the second, mark the first as spam, and archive the others.

```
; M r 200712/3
; M s 200712/4/text
; M a
```

Prepare to use the script `M` (like above) but only for messages from december 2007 that contain PDF attachments and are kept in the omsgs mailbox:

```
; ls /mail/box/nemo/msgs/200712/*//*.pdf >/tmp/msgs.nemo
```

Use `mailplumb` to see in faces messages in the Plan B mailbox:

```
plumber
mailplumb
```

```
faces -m /mail/box/$user/msgs
```

This is a guide for reading mail using `olive(1):

```
!Msgs       # ask for mail index
!Arch       # archive this mail
!Spam       # mark this mail as spam
X/text/D    # delete all panels showing msgs
```
, <Msgs  # update mail index
,  >Arch  # archive all msgs listed
,  x/9fans/+−p  # list all 9fans messages shown

The script /sys/src/cmd/mail2fs/Month is an example of a per-month cleanup script. Usually the directory for the last month is declared as DMTEMP and this permission is cleared when all spam has been dealt with. The script also creates cached listings for all but the current month.

Add msgs from the UNIX-format oldmbox to the standard msgs folder, honoring their dates and inserting them as archived, and produce a listing for them kept at newfolder.

; mail2fs −aon oldmbox > /tmp/archived
; mlist </tmp/archived >newfolder

Create a cache for September, 2009, to be used by tools listing msgs.

; cd /mail/box/$usermsgs
; msgs −a msg 200909 > 200909.l

FILES
/mail/box/$user/mbox  Standard Plan 9 mail box.
/mail/box/$usermsgs/  Standard Plan B mail box.
/tmp/msgs.$$user  List of msgs being processed by the user.

SOURCE
/sys/src/cmd/upas/mail2fs and /acme/msgs

SEE ALSO
mail(1).
NAME
  mtime - print file modification time

SYNOPSIS
  mtime file ...

DESCRIPTION
  Mtime prints the modification time (in seconds since the epoch) and name of each file.

SOURCE
  /sys/src/cmd/mtime.c
NAME
mug – convert an image to a face icon

SYNOPSIS
mug [ file ]

DESCRIPTION
Mug reads a Plan 9 image(6) from file (or standard input if there is no file) and displays a working
version of the icon a gray ramp, and a larger image (the ‘crop box’), all derived from file. Selecting
Write from the button–3 menu will write the icon in face(6) format to standard output.

Imagine a 3x3 grid on the crop box. You can move an edge or corner of the box by putting the
mouse in the corresponding section of the grid and dragging. Dragging in the middle box in the
grid translates the crop box. The mouse cursor changes to tell you where you are.

The bar in the gray ramp controls the map from picture gray levels to the output levels. The val-
ues along the bar are mapped to 0 through 255 in the output. You can move the bar vertically by
grabbing the midsection or adjust the width by grabbing an endpoint.

The current icon is shown in the bottom left corner, surrounded by eight small empty boxes. You
can save the settings as they are by dragging the current icon into one of the other boxes. You
can restore the settings by dragging an icon from one of the periphery boxes into the middle.

EXAMPLES
Convert a JPEG image into a face icon.

    jpg −c plus.jpg | mug >plus.1

SEE ALSO
faces(1), jpg(1), face(6), image(6)
NAME
nedmail - reading mail

SYNOPSIS
upas/nedmail[−nr][−f mailfile][−s mailfile]
upas/nedmail−c dir

DESCRIPTION
Nedmail edits a mailbox. The default mailbox is /mail/box/username/mbox. The −f command line option specifies an alternate mailbox. Unrooted path names are interpreted relative to /mail/box/username. If the mailfile argument is omitted, the name defaults to stored.

The options are:
−c dir Create a mailbox. If dir is specified, the new mailbox is created in /mail/box/username/dir/mbox. Otherwise, the default mailbox is created.
−r Reverse: show messages in first-in, first-out order; the default is last-in, first-out.
−n Make the message numbers the same as the file names in the mail box directory. This implies the −r option.
−f mailfile Read messages from the specified file (see above) instead of the default mailbox.
−s mailfile Read a single message file mailfile, as produced by fs, and treat it as an entire mailbox. This is provided for use in plumbing rules; see faces(1).

Nedmail starts by reading the mail box, printing out the number of messages, and then prompting for commands from standard input. Commands, as in ed(1), are of the form [\range] command [arguments]. The command is applied to each message in the (optional) range.

The address range can be:
address to indicate a single message header
address address to indicate a range of contiguous message headers
g/ expression/ to indicate all messages whose headers match the regular expression.
g% expression% to indicate all messages whose contents match the regular expression.

The addresses can be:
number to indicate a particular message
address . number to indicate a subpart of a particular message
/ expression/ to indicate the next message whose header matches expression
% expression% to indicate the next message whose contents match expression
empty or . to indicate the current message
− address to indicate backwards search or movement

Since messages in MIME are hierarchical structures, in nedmail all the subparts are individually addressable. For example if message 2 contains 3 attachments, the attachments are numbered 2.1, 2.2, and 2.3.

The commands are:
a args Reply to all addresses in the To:, From:, and Cc: header lines. Marshal is used to format the reply and any arguments the user specifies are added to the command line to marshal before the recipient. The possibility of making a fool of yourself is very high with this command.
A args Like a but with the message appended to the reply.
b Print the headers for the next ten messages.
d Mark message to be deleted upon exiting nedmail.
f Append the message to the file /mail/box/username/sendername where sendername is the account name of the sender.
h Print the disposition, size in characters, reception time, sender, and subject of the message.
H Print the MIME structure of the message.
help            Print a summary of the commands.
  m person ...   Forward the message as a mime attachment to the named persons.
  M person ...   Like m but allow the user to type in text to be included with the forwarded message.
  p              Print message. An interrupt stops the printing.
  r args        Reply to the sender of the message. Marshal is used to format the reply. If and optional Args are specified, they are added to the command line to marshal before the recipient’s address.
  R args        Like r but with the original message included as an attachment.
  rf            Like R but append the message and the reply to the file
                 /mail/box/username/sendername where sendername is the account name of the sender.
  Rf            Like R but append the message and the reply to the file
                 /mail/box/username/sendername where sendername is the account name of the sender.
  s mfile       Append the message to the specified mailbox. If mfile doesn’t start with a ’/’, it is interpreted relative to the directory in which the mailbox resides. If mfile is a directory then the destination is a file in that directory. If the MIME header specifies a file name, that one is used. Otherwise, one is generated using mktemp(2) and the string att.XXXXXXXXX.
  q              Put undeleted mail back in the mailbox and stop.
  EOT (control-D) Same as q.
  w file        Same as s with the mail header line(s) stripped. This can be used to save binary mail bodies.
  u              Remove mark for deletion.
  x              Exit, without changing the mailbox file.
  y              Synchronize with the mail box. Any deleted messages are purged and any new messages read. This is equivalent to quitting nedmail and restarting.
  | command      Run the command with the message body as standard input.
  !command      Run the command with the whole message as standard input.
  =              Escape to the shell to do command.

Here’s an example of a mail session that looks at a summary of the mail messages, saves away an html file added as an attachment to a message and then deletes the message:

```
% mail
7 messages
: ,h
1 H 2129 07/22 12:30 noone@madeup.net "Add Up To 2000 free miles"
2 504 07/22 11:43 jmk
3 H 784 07/20 09:05 presotto
4 822 07/11 09:23 xxx@yyy.net "You don’t call, you don’t write..."
5 193 07/06 16:55 presotto
6 529 06/01 19:42 jmk
7 798 09/02 2000 howard
: 1H
1 multipart/mixed 2129 from=noone@madeup.net
  1.1 text/plain 115
  1.2 text/html 1705 filename=northwest.htm
: 1.2w /tmp/northwest.html
!saved in /tmp/northwest.html
1.2: d
1: q
!1 message deleted
%
```

Notice that the delete of message 1.2 deleted the entire message and not just the attachment.

**FILES**

```
/mail/box/* mail directories
```
/mail/box/*/mbox          mailbox files
/mail/box/*/forward      forwarding address(es)
/mail/box/*/pipeto       mail filter
/mail/box/*/L.reading    mutual exclusion lock for multiple mbox readers
/mail/box/*/L.mbox       mutual exclusion lock for altering mbox

SOURCE
/sys/src/cmd/upas/ned

SEE ALSO
    mail(1), aliasmail(8), filter(1), marshal(1), mlmgr(1), nedmail(1), upasfs(4), smtp(8), faces(1), rewrite(6)
NAME
  netstat – summarize network connections

SYNOPSIS
  netstat [−in][−p proto][ netmtpt ]

DESCRIPTION
  Netstat prints information about network mounted at netmtpt, default /net. For IP connections,
  netstat reports the protocol, connection number, user, connection state, local port, remote port
  and remote address.

  The options are:
  −i  Instead of the usual listing, print one line per network interface. Each line gives the device,
       MTU, local address, mask, remote address, packets in, packets out, errors in, and errors
       out for this interface.
  −n  By default, netstat looks up port numbers and addresses in the network databases to print
       symbolic names if possible. This option disables such translation.
  −p  Show only connections with the given protocol.

FILES
  /net/*/ *

SOURCE
  /sys/src/cmd/netstat.c

SEE ALSO
  ipconfig(8)
NAME
   news - print news items

SYNOPSIS
   news [−a][−n][ item ...]

DESCRIPTION
   When invoked without options, this simple local news service prints files that have appeared in
   /lib/news since last reading, most recent first, with each preceded by an appropriate header.
   The time of reading is recorded. The options are
   −a   Print all items, regardless of currency. The recorded time is not changed.
   −n   Report the names of the current items without printing their contents, and without chang-
        ing the recorded time.

   Other arguments select particular news items.
   To post a news item, create a file in /lib/news.

   You may arrange to receive news automatically by registering your mail address in
   /sys/lib/subscribers. A daemon mails recent news to all addresses on the list.

   Empty news items, and news items named core or dead.letter are ignored.

FILES
   /lib/news/*        articles
   $HOME/lib/newstime modify time is time news was last read
   /sys/lib/subscribers who gets news mailed to them

SOURCE
   /sys/src/cmd/news.c
NAME
   nm - name list (symbol table)

SYNOPSIS
   nm [-aghnsTu] file ...

DESCRIPTION
   Nm prints the name list of each executable or object file in the argument list. If the file is an
   archive (see ar(1)), the name list of each file in the archive is printed. If more than one file is given
   in the argument list, the name of each file is printed at the beginning of each line.

   Each symbol name is preceded by its hexadecimal value (blanks if undefined) and one of the let-
   ters
       T  text segment symbol
       t  static text segment symbol
       L  leaf function text segment symbol
       l  static leaf function text segment symbol
       D  data segment symbol
       d  static data segment symbol
       B  bss segment symbol
       b  static bss segment symbol
       a  automatic (local) variable symbol
       p  function parameter symbol
       z  source file name
       Z  source file line offset
       f  source file name components

   The output is sorted alphabetically.

   Options are:
   -a  Print all symbols; normally only user–defined text, data, and bss segment symbols are
        printed.
   -g  Print only global (T, L, D, B) symbols.
   -h  Do not print file name headers with output lines.
   -n  Sort according to the address of the symbols.
   -s  Don’t sort; print in symbol–table order.
   -T  Prefix each line with the symbol’s type signature.
   -u  Print only undefined symbols.

SOURCE
   /sys/src/cmd/nm.c

SEE ALSO
   ar(1), 2l(1), db(1), acid(1), a.out(6)
NAME
ns – display name space

SYNOPSIS
ns [−r][ pid ]

DESCRIPTION
Ns prints a representation of the file name space of the process with the named pid, or by default itself. The output is in the form of an rc(1) script that could, in principle, recreate the name space. The output is produced by reading and reformatting the contents of /proc/pid/ns.

By default, ns rewrites the names of network data files to represent the network address that data file is connected to, for example replacing /net/tcp/82/data with tcp!123.122.121.9. The −r flag suppresses this rewriting.

FILES
/proc/*/ns

SOURCE
/sys/src/cmd/ns.c

SEE ALSO
ps(1), proc(3), namespace(4), namespace(6)

BUGS
The names of files printed by ns will be inaccurate if a file or directory it includes has been renamed.
NAME
   p - paginate

SYNOPSIS
   p [ −number ] [ file ... ]

DESCRIPTION
   P copies its standard input, or the named files if given, to its standard output, stopping at the end of every 22nd line, and between files, to wait for a newline from the user. The option sets the number of lines on a page.

   While waiting for a newline, p interprets the commands:
   ! Pass the rest of the line to the shell as a command.
   q Quit.

SOURCE
   /sys/src/cmd/p.c
NAME
page – view FAX, image, graphic, PostScript, PDF, and typesetter output files

SYNOPSIS
page [-abirPRVw][ -p ppi] [file...]

DESCRIPTION
Page is a general purpose document viewer. It can be used to display the individual pages of a
PostScript, PDF, or tex(1) or troff(1) device independent output file. Tex or troff output is simply
converted to PostScript in order to be viewed. It can also be used to view any number of graphics
files (such as a FAX page, a Plan 9 image(6) file, an Inferno bitmap file, or other common format).
Page displays these in sequence. In the absence of named files, page reads one from standard
input.

By default, page runs in the window in which it is started and leaves the window unchanged. The
-R option causes page to grow the window if necessary to display the page being viewed. The -w
option causes page to create a new window for itself. The newly created window will grow as
under the -R option. If being used to display multipage documents, only one file may be specified
on the command line.

The -p option sets the resolution for PostScript and PDF files, in pixels per inch. The default is
100 ppi. The -a option reverses the order in which pages are displayed.

When viewing a document, page will try to guess the true bounding box, usually rounding up from
the file's bounding box to 8½×11 or A4 size. The -b option causes it to respect the bounding
box given in the file. As a more general problem, some PostScript files claim to conform to
Adobe's Document Structuring Conventions but do not. The -P option enables a slightly slower
and slightly more skeptical version of the PostScript processing code. Unfortunately, there are
PostScript documents that can only be viewed with the -P option, and there are PostScript docu-
ments that can only be viewed without it.

When viewing images with page, it listens to the image plumbing channel (see plumber(4)) for
more images to display. The -i option causes page to not load any graphics files nor to read
from standard input but rather to listen for ones to load from the plumbing channel.

The -v option turns on extra debugging output, and the -V option turns on even more debugging
output. The -a option causes page to call abort(2) rather than exit cleanly on errors, to facilitate
debugging.

Pressing and holding button 1 permits panning about the page.

Button 2 raises a menu of operations on the current image or the entire set. The image transfor-
mations are non-destructive and are valid only for the currently displayed image. They are lost as
soon as another image is displayed. The button 2 menu operations are:

Orig size
Restores the image to the original. All modifications are lost.

Zoom
Prompts the user to sweep a rectangle on the image which is expanded proportionally
to the rectangle.

Fit window
Resizes the image so that it fits in the current window.

Rotate 90
Rotates the image 90 degrees clockwise

Upside down
Toggles whether images are displayed upside-down.

Next
Displays the next page.

Prev
Displays the previous page.

Zerox
Displays the current image in a new page window. Useful for selecting important pages
from large documents.

Reverse
Reverses the order in which pages are displayed.

Write
Writes the image to file.

Button 3 raises a menu of the pages to be selected for viewing in any order.
Typing a q or control-D exits the program. Typing a u toggles whether images are displayed upside-down. (This is useful in the common case of mistransmitted upside-down faxes). Typing a r reverses the order in which pages are displayed. Typing a w will write the currently viewed page to a new file as a compressed image file. When possible, the filename is of the form basename.pagenum.bit. Typing a d removes an image from the working set.

To go to a specific page, one can type its number followed by enter. Typing left arrow, backspace, or minus displays the previous page. Typing right arrow, space, or enter displays the next page. The up and down arrow pan up and down one half screen height, changing pages when panning off the top or bottom of the page.

Page calls gs(1) to draw each page of PostScript and PDF files. It also calls a variety of conversion programs, such as those described in jpg(1), to convert the various raster graphics formats into Inferno bitmap files. Pages are converted “on the fly,” as needed.

EXAMPLES
page /sys/src/cmd/gs/examples/tiger.eps
    Display a color PostScript file.
page /usr/inferno/icons/*.bit
    Browse the Inferno bitmap library.
man −t page | page −w
    Preview this manual in a new window.

SEE ALSO
    gs(1), jpg(1), tex(1), traff(1)

SOURCE
    /sys/src/cmd/page

DIAGNOSTICS
    The mouse cursor changes to an arrow and ellipsis when page is reading or writing a file.

BUGS
    Page supports reading of only one document file at a time, and the user interface is clumsy when viewing very large documents.

    When viewing multipage PostScript files that do not contain “%%Page” comments, the button 3 menu only contains “this page” and “next page”: correctly determining page boundaries in Postscript code is not computable in the general case.

    If page has trouble viewing a Postscript file, it might not be exactly conforming: try viewing it with the −P option.

    The interface to the plumber is unsatisfactory. In particular, document references cannot be sent via plumbing messages.

    There are too many keyboard commands.
NAME
   passwd, netkey - change or verify user password

SYNOPSIS
   passwd [ username[@domain] ]
   netkey

DESCRIPTION
   *Passwd* changes the invoker's Plan 9 password and/or APOP secret. The Plan 9 password is used to login to a terminal while the APOP secret is used for a number of external services: POP3, IMAP, and VPN access. The optional argument specifies the user name and authentication domain to use if different than the one associated with the machine *passwd* is run on.

   The program first prompts for the old Plan 9 password in the specified domain to establish identity. It then prompts for changes to the password and the secret. New passwords and secrets must be typed twice, to forestall mistakes. New passwords must be sufficiently hard to guess. They may be of any length greater than seven characters.

   *Netkey* prompts for a password to encrypt network challenges. It is a substitute for a SecureNet box.

   These commands may be run only on a terminal, to avoid transmitting clear text passwords over the network.

SOURCE
   /sys/src/cmd/auth/passwd.c
   /sys/src/cmd/auth/netkey.c

SEE ALSO
   readnram in authsrv(2), encrypt(2), cons(3), auth(8), securenet(8)


BUGS
   Now that *cpu* connections are always encrypted, the only good reason to require that these commands be run only on terminals is concern that the CPU server might be subverted.
NAME
patch - simple patch creation and tracking system

SYNOPSIS
patch/create name email files ... [< description ]
patch/list [ name ... ]
patch/diff name
patch/apply name
patch/undo name
patch/note name [ < note ]

DESCRIPTION
These scripts are a simple patch submission and tracking system used to propose additions or changes to Plan 9. There is no guarantee that any patch will be accepted, nor that it will be accepted verbatim. Each patch has a name (lowercase letters, numbers, dash, dot, and underscore only) and is stored in /n/sources/patch/name.

Patch/create creates a new patch consisting of the changes to the listed files from the distribution, reading a description of the patch from standard input: please provide an explanation of what the change is supposed to do, some context, and a rationale for the change. Test data or pointers to same to verify that the fix works are also welcome. When sending a patch, follow these guidelines:

• Before preparing the patch, run replica/pull and base your patch on current distribution source code.
• If this is a bug fix, explain the bug clearly. Don’t assume the bug is obvious from the fix.
• If this is a new feature, explain it clearly. Don’t assume it is obvious from the change.
• Make the new code look as much like the old code as possible: don’t make gratuitous changes, and do follow the style of the old code. See style(6) for the canonical Plan 9 coding style.
• If your patch changes externally-visible behaviour, update the manual page.

The email address, if not --, will be sent notification messages when the patch is applied, rejected, or commented on. If rejected, the e-mail will contain a note explaining why and probably listing suggested changes and encouraging you to resubmit.

Patch/list displays information about the named patches, or all currently pending patches if none are specified.

Patch/diff shows a patch as diffs between the original source files and the patched source files.

Patch/apply applies the patch to the current source tree. It is intended to be run by the Plan 9 developers with pie as their root file system. If the source has changed since the patch was created, apply will report the conflict and not change any files. Before changing any files, patch/apply makes backup copies of the current source tree’s files. The backups are stored in the patch directory.

Patch/undo will copy the backups saved by patch/apply back into the source tree. It will not restore a backup if the file being replaced is not byte-identical to the one created by patch/apply.

EXAMPLES
Propose a change to pwd, which you have modified locally:

% patch/create pwd-errors user@host.dom /sys/src/cmd/pwd.c
Fix pwd to print errors to fd 2 rather than 1.
^D
%

Then the developers at Bell Labs run

patch/diff pwd-errors

to inspect the change (possibly viewing /n/sources/patch/pwd-errors/pwd.c to see the larger context). To make the change, they run
patch/apply pwd-errors

Otherwise they run

% patch/note pwd-errors
Pwd should definitely print errors to fd 1 because ...
^D
%

to add a note to the /n/sources/pwd-errors/notes file.

FILES
/n/sources/patch

SOURCE
/rc/bin/patch

SEE ALSO
diff(1)
NAME
   pcc - APE C compiler driver

SYNOPSIS
   pcc [ option ... ] [ name ... ]

DESCRIPTION
   Pcc compiles and loads C programs, using APE (ANSI C/POSIX) include files and libraries. Named files ending with .c are preprocessed with cpp(1), then compiled with one of the compilers described in 2c(1), as specified by the environment variable $objtype. The object files are then loaded using one of the loaders described in 2l(1). The options are:
   
   - + Accept C++ // comments.
   - o out Place loader output in file out instead of the default 2 . out, v . out, etc.
   - P Omit the compilation and loading phases; leave the result of preprocessing name . c in name . i.
   - E Like -P, but send the result to standard output.
   - c Omit the loading phase.
   - p Insert profiling code into the executable output.
   - w Print compiler warning messages.
   - 1 lib Include / $objtype / lib / ape / lib a as a library during the linking phase.
   - B Don't complain about functions used without ANSI function prototypes.
   - v Enable void* conversion warnings, as in 2c(1).
   - v Echo the preprocessing, compiling, and loading commands before they are executed.
   - D name=def Define the name to the preprocessor, as if by #define. If no definition is given, the name is defined as 1.
   - U name Undefine the name to the preprocessor, as if by #undef.
   - I dir #include files whose names do not begin with / are always sought first in the directory of the file argument, then in directories named in - I options, then in / $objtype / include / ape.
   - N Don't optimize compiled code.
   - S Print an assembly language version of the object code on standard output.
   - a Instead of compiling, print on standard output acid functions (see acid(1)) for examining structures declared in the source files.
   - aa Like - a except that functions for structures declared in included header files are omitted.
   - F Enable vararg type checking as described in 2c(1). This is of limited use without the appropriate #pragma definitions.

The APE environment contains all of the include files and library routines specified in the ANSI C standard (X3.159–1989), as well as those specified in the IEEE Portable Operating System Interface standard (POSIX, 1003.1–1990, ISO 9945–1). In order to access the POSIX routines, source programs should define the preprocessor constant _POSIX_SOURCE.

FILES
   /sys / include / ape directory for machine-independent #include files.
   / $objtype / include / ape directory for machine-dependent #include files.
   / $objtype / lib / ape / libap.a ANSI C/POSIX library.

SEE ALSO
   cpp(1), 2c(1), 2a(1), 2l(1), mk(1), nm(1), acid(1), db(1), prof(1)
Howard Trickey, “APE — The ANSI/POSIX Environment”

SOURCE
/sys/src/cmd/pcc.c

BUGS
The locale manipulation functions are minimal. Signal functions and terminal characteristic handlers are only minimally implemented. Link always fails, because Plan 9 doesn’t support multiple links to a file. The functions related to setting effective user and group ids cannot be implemented because the concept doesn’t exist in Plan 9.
NAME
pic, tpic - troff and tex preprocessors for drawing pictures

SYNOPSIS
pic [files]
tpic [files]

DESCRIPTION
Pic is a troff(1) preprocessor for drawing figures on a typesetter. Pic code is contained between .PS and .PE lines:

```
. PS optional-width optional-height
   element-list
. PE
```
or in a file mentioned in a .PS line:

```
. PS <file
```
If optional-width is present, the picture is made that many inches wide, regardless of any dimensions used internally. The height is scaled in the same proportion unless optional-height is present. If .PF is used instead of .PE, the typesetting position after printing is restored to what it was upon entry.

An element-list is a list of elements:

```
primitive attribute-list
placename : element
placename : position
var = expr
direction
{ element-list }
[ element-list ]
for var = expr to expr by expr do { anything }
if expr then { anything } else { anything }
copy file, copy thru macro, copy file thru macro
sh { commandline }
print expr
reset optional var-list
troff-command
```

Elements are separated by newlines or semicolons; a long element may be continued by ending the line with a backslash. Comments are introduced by a # and terminated by a newline. Variable names begin with a lower case letter; place names begin with upper case. Place and variable names retain their values from one picture to the next.

After each primitive the current position moves in the current direction (up, down, left, right (default)) by the size of the primitive. The current position and direction are saved upon entry to a {...} block and restored upon exit. Elements within a block enclosed in [...] are treated as a unit; the dimensions are determined by the extreme points of the contained objects. Names, variables, and direction of motion within a block are local to that block.

Troff-command is any line that begins with a period. Such a line is assumed to make sense in the context where it appears; generally, this means only size and font changes.

The primitive objects are:
```
box circle ellipse arc line arrow spline move text-list
```
arrow is a synonym for line ->.

An attribute-list is a sequence of zero or more attributes; each attribute consists of a keyword, perhaps followed by a value.

```
h(eight) expr   wid( th) expr
rad(ius) expr   diam(eter) expr
up opt-expr     down opt-expr
right opt-expr  left opt-expr
from position   to position
```

---

PIC(1)

164
at position with corner
by expr, expr then
dotted opt-expr dashed opt-expr
chop opt-expr \rightarrow < \leftarrow
invis same
fill opt-expr

Missing attributes and values are filled in from defaults. Not all attributes make sense for all primitives; irrelevant ones are silently ignored. The attribute at causes the geometrical center to be put at the specified place; with causes the position on the object to be put at the specified place. For lines, splines and arcs, height and width refer to arrowhead size. A bare expr implies motion in the current direction.

Text is normally an attribute of some primitive; by default it is placed at the geometrical center of the object. Stand-alone text is also permitted. A text list is a list of text items:

text-item:
"..." positioning ...
sprintf("format", expr, ...) positioning ...

positioning:
center ljust rjust above below
If there are multiple text items for some primitive, they are arranged vertically and centered except as qualified. Positioning requests apply to each item independently. Text items may contain troff commands for size and font changes, local motions, etc., but make sure that these are balanced so that the entering state is restored before exiting.

A position is ultimately an x,y coordinate pair, but it may be specified in other ways.

position:
expr, expr
place \pm expr, expr
place \pm ( expr, expr )
( position, position ) x from one, y the other
expr[of the way] between position and position
expr < position, position >
( position )

place:
placename optional-corner
corner of placename
nth primitive optional-corner
corner of nth primitive

Here

An optional-corner is one of the eight compass points or the center or the start or end of a primitive.

optional-corner:
.n .e .w .s .ne .se .nw .sw .c .start .end
corner:
top bot left right start end

Each object in a picture has an ordinal number; nth refers to this.

nth:
nth, nth last

The built-in variables and their default values are:

boxwid 0.75  boxht 0.5
circlerad 0.25 arcrad 0.25
ellipsewid 0.75  ellipsetht 0.5
linewid 0.5   lineht 0.5
movewid 0.5   moveht 0.5
textwid 0     textht 0
arrowwid 0.05 arrowht 0.1
dashwid 0.1   arrowhead 2
scale 1

These may be changed at any time, and the new values remain in force from picture to picture
until changed again or reset by a reset statement. Variables changed within [ and ] revert to their previous value upon exit from the block. Dimensions are divided by scale during output.

Expressions in pic are evaluated in floating point. All numbers representing dimensions are taken to be in inches. 

expr:

```
expr op expr
- expr
! expr
( expr )
variable
number
place .x place .y place .ht place .wid place .rad
sin(expr) cos(expr) atan2(expr,expr) log(expr) exp(expr)
sqrt(expr) max(expr,expr) min(expr,expr) int(expr) rand()
```

op:

```
+ - * / % < <= > >= == != && ||
```

The define and undef statements are not part of the grammar.

```
define name { replacement text }
define name
```

Occurrences of $1, $2, etc., in the replacement text will be replaced by the corresponding arguments if name is invoked as

```
name(arg1, arg2, ...)
```

Non-existent arguments are replaced by null strings. Replacement text may contain newlines. The undef statement removes the definition of a macro.

Tpic is a tex(1) preprocessor that accepts pic language. It produces Tex commands that define a box called \graph, which contains the picture. The box may be output this way:

```
\centerline{\box\graph}
```

EXAMPLES

```
arrow "input" above; box "process"; arrow "output" above move
A: ellipse
circle rad .1 with .w at A.e
circle rad .05 at 0.5 <A.c, A.ne>
circle rad .065 at 0.5 <A.c, A.ne>
spline from last circle.nw left .25 then left .05 down .05 arc from A.c to A.se rad 0.5
for i = 1 to 10 do { line from A.s+.025*i,.01*i down i/50 }
```

SOURCE

```
/sys/src/cmd/pic
```

SEE ALSO

```
grap(1), doctype(1), troff(1)
```

NAME
pipefile - attach filter to file in name space

SYNOPSIS
pipefile [−d][−r command][−w command] file

DESCRIPTION
Pipefile uses bind(2) to attach a pair of pipes to file, using them to interpose filter commands between the true file and the simulated file that subsequently appears in the name space. Option −r interposes a filter that will affect the data delivered to programs that read from file; −w interposes a filter that will affect the data written by programs to file. At least one command must be specified; pipefile will insert a cat(1) process in the other direction.

After pipefile has been run, the filters are established for programs that subsequently open the file; programs already using the file are unaffected.

Pipefile opens the file twice, once for each direction. If the file is a single-use device, such as /dev/mouse, use the −d flag to specify that the file is to be opened once, in ORDWR mode.

EXAMPLES
Simulate an old terminal:
  % pipefile −w 'tr a−z A−Z' /dev/cons
  % echo hello
  HELLO
  %

Really simulate an old terminal:
  % pipefile −r 'tr A−Z a−z' −w 'tr a−z A−Z' /dev/cons
  % echo hello
  HELLO
  %

SOURCE
/sys/src/cmd/pipefile.c

SEE ALSO
mouse(8)

BUGS
The I/O model of pipefile is peculiar; it doesn’t work well on plain files. It is really intended for use with continuous devices such as /dev/cons and /dev/mouse. Pipefile should be rewritten to be a user−level file system.

If the program using the file managed by pipefile exits, the filter will see EOF and exit, and the file will be unusable until the name space is repaired.
NAME
plot - graphics filter
SYNOPSIS
plot [ file ... ]
DESCRIPTION
Plot interprets plotting instructions (see plot(6)) from the files or standard input, drawing the results in a newly created rio(1) window. Plot persists until a newline is typed in the window. Various options may be interspersed with the file arguments; they take effect at the given point in processing. Options are:

-d Double buffer: accumulate the plot off-screen and write to the screen all at once when an erase command is encountered or at end of file.
-e Erase the screen.
-c col Set the foreground color (see plot(6) for color names).
-£ fill Set the background color.
-g grade Set the quality factor for arcs. Higher grades give better quality.
-p col Set the pen color.
-w Pause until a newline is typed on standard input.
-C Close the current plot.
-W x0,y0,x1,y1 Specify the bounding rectangle of plot's window. By default it uses a 512×512 window in the middle of the screen.

SOURCE
/sys/src/cmd/plot
SEE ALSO
rio(1), plot(6)
NAME
plumb - send message to plumber

SYNOPSIS
plumb [−p plumbfile][−a attributes][−s source][−d destination][−t type][−w directory]
−i | data...

DESCRIPTION
The plumb command formats and sends a plumbing message whose data is, by default, the con-
catenation of the argument strings separated by blanks. The options are:
−p write the message to plumbfile (default /mnt/plumb/send).
−a set the attr field of the message (default is empty).
−s set the src field of the message (default is plumb).
−d set the dst field of the message (default is empty).
−t set the type field of the message (default is text).
−w set the wdir field of the message (default is the current working directory of plumb).
−i take the data from standard input rather than the argument strings. If an action=
attribute is not otherwise specified, plumb will add an action=showdata attribute to
the message.

FILES
/usr/$user/lib/plumbing default rules file
/mnt/plumb mount point for plumber(4).

SOURCE
/sys/src/cmd/plumb

SEE ALSO
plumb(2), plumber(4), plumb(6)
NAME
pq, pqgen, pqsrv – query POST database

SYNOPSIS
pq [-lf] [-d debug] [-m modules] [-o format] query ...
pqgen [ directory ]
pqsrv [ modules ]

DESCRIPTION
Pq connects to an Implicit Relational Database (IRDB) directory (a read–only, text–based relational database) and outputs the record(s) that match the query arguments. The directory contacted may be specified by a modules argument (see dispatch(7)). By default it is the corporate LUCID directory, which is a superset of the corporate POST directory.

Each query argument results in an independent query of the directory, consisting of a set of input attributes, obtained from the query argument, and a set of output attributes, obtained from the output format (see the −o option). The output of all the queries are formatted according to the output format and concatenated on standard output.

Each query argument is a list of strings of the form attribute=value separated by any number of separator characters (|). It is an error if there is no corresponding attribute name. The query argument may be the empty string; in this case, all records that contain the output attributes are returned. The options are:

−d debug
Print information regarding internal operations. Values for debug range from 1 to 3, with 3 yielding the most detail.

−f
Changes the default output format to a more verbose (‘full’) one.

−l
is similar but omits the html attribute and consults only POST, not LUCID. This seems to be necessary to find ex–Alcatel employees.

−m modules
Contact the directory specified by the modules string; the format is described in dispatch(7). The default value for modules is the empty string.

−o format
Use the output format specified. See below for a full description.

Output Formatting
The format argument specified with the −o option is used like a print(2) string for formatting the output of directory queries. All characters are copied literally, except for attribute substitutions and backslash escapes. Quoting may be necessary to prevent shell interpretation. The syntax for attribute substitutions is this:

% flags minimum . maximum [attribute]

Only % and attribute are required. Curly braces, {}, are required only when the attribute is immediately followed by an alphanumeric. Flags may be one or more of the following:

− Right justify (left justification is the default).
^ Capitalize the first letter of each word.
+ Capitalize all letters in the value.
< If the value is empty, delete back to the last \< or beginning of output.
> If the value is empty, skip to the next \> or end of format string.

Minimum is an integer giving the minimum field width. If the value has fewer than the minimum number of characters, the field will be padded with blanks. The default minimum is zero.

Maximum is an integer that specifies the maximum number of characters to be output from the value. If the value has more characters than this number, the value will be truncated. A maximum of zero (the default) causes all characters to be output. A period is used to separate minimum and maximum and is only required if maximum is specified.
The following table lists backslash escapes that are recognized by *pq*(1):
center,tab(); c l. *Escape; Meaning*

\b:Backspace \c:Suppress terminating newline \f:Formfeed \n:Newline \oo:ASCII character defined by an octal number \r:Carriage return \t:Tab \v:Vertical tab \:<;Marker for < flag
\>:Marker for > flag

*Pqgen* is used to create index files for an existing IRDB *ev*(7) directory. Once indexed, the speed of lookups is greatly improved. However, it is then necessary to rerun *pqgen* after any changes are made to the *ev* database. The *directory* is the location of the *ev* database. If not an absolute pathname, it is interpreted relative to /lib/pq.

*Pqsrv* is the server that handles incoming PQ requests. It is meant to be run by *listen*(8), typically for TCP port 411. The optional *modules* argument is the same as that to *pq*(1) above.

**EXAMPLES**
Find the telephone number of employee with login of liz:

```
pq −o %telephone 'login=liz'
```

List addresses of employees in New Jersey and Texas who are full-time:

```
pq −o '%20name %25addr %state %zip' 'status=FT|state=NJ|state=TX'
```

Consult /sys/src/cmd/pq/example for a more detailed example.

**FILES**

```
//lib/pq/dispatch default dispatch file
/rc/bin/service/tcp411 typical location for pqsrv
```

**SOURCE**

```
/sys/src/cmd/pq
/sys/src/libpq
```

**SEE ALSO**

*listen*(8)

**DIAGNOSTICS**

*Pq* prints a diagnostic to the standard error and exits with status no records if there are no matches. Otherwise, it returns a successful status.
NAME
   pr - print file

SYNOPSIS
   pr [ option ... ] [ file ... ]

DESCRIPTION
   Pr produces a printed listing of one or more files on its standard output. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. With no file arguments, pr prints its standard input.

   Options apply to all following files but may be reset between files:
   
   -n   Produce n-column output.
   +n   Begin printing with page n.
   -b   Balance columns on last page, in case of multi-column output.
   -d   Double space.
   -en  Set the tab stops for input text every n spaces.
   -h   Take the next argument as a page header (file by default).
   -in  Replace sequences of blanks in the output by tabs, using tab stops set every n spaces.
   -f   Use form feeds to separate pages.
   -ln  Take the length of the page to be n lines instead of the default 66.
   -m   Print all files simultaneously, each in one column.
   -nm  Number the lines of each file. The numeric argument m, default 5, sets the width of the line-number field.
   -on  Offset the left margin n character positions.
   -p   Pad each file printed to an even number of pages, if necessary. For two-sided printers, this will ensure each file will start a new page.
   -sc  Separate columns by the single character c instead of aligning them with white space. A missing c is taken to be a tab.
   -t   Do not print the 5-line header or the 5-line trailer normally supplied for each page.
   -wn  For multi-column output, take the width of the page to be n characters instead of the default 72.

SOURCE
   /sys/src/cmd/pr.c

SEE ALSO
   cat(1), lp(1)
**NAME**

prof, tprof, kprof – display profiling data

**SYNOPSIS**

```bash
prof [-dr] [program] [profile]
```

```bash
tprof pid
```

```bash
kprof kernel kpdata
```

**DESCRIPTION**

`prof` interprets files produced automatically by programs loaded using the `-p` option of `2l(1)` or other loader. The symbol table in the named program file (2.out etc., according to `$objtype`, by default) is read and correlated with the profile file (`prof.out` by default). For each symbol, the percentage of time (in seconds) spent executing between that symbol and the next is printed (in decreasing order), together with the time spent there and the number of times that routine was called.

Under option `-d`, `prof` prints the dynamic call graph of the target program, annotating the calls with the time spent in each routine and those it calls, recursively. The output is indented two spaces for each call, and is formatted as

```
symbol:time/ncall
```

where `symbol` is the entry point of the call, `time` is in milliseconds, and `ncall` is the number of times that entry point was called at that point in the call graph. If `ncall` is one, the `/ncall` is elided. Normally recursive calls are compressed to keep the output brief; option `-r` prints the full call graph.

The size of the buffer in `program` used to hold the profiling data, by default 2000 entries, may be controlled by setting the environment variable `profsize` before running `program`. If the buffer fills, subsequent function calls may not be recorded.

The profiling code provided by the linker initializes itself to profile the current pid, producing a file called `prof.pid`. If a process forks, only the parent will continue to be profiled. Forked children can cause themselves to be profile by calling

```
prof(fn, arg, entries, what)
```

which causes the function `fn(arg)` to be profiled. When `fn` returns `prof.pid` is produced for the current process pid.

The environment variable `proftype` can be set to one of `user`, `kernel`, `elapsed`, or `sample`, to profile time measured spent in user mode, time spent in user+kernel mode, or elapsed time, using the cycle counter, or the time in user mode using the kernel’s HZ clock. The cycle counter is currently only available on modern PCs and on the PowerPC. Default profiling measures user time, using the cycle counter if it is available.

`tprof` is similar to `prof`, but is intended for profiling multiprocess programs. It uses the `/proc/pid/profile` file to collect instruction frequency counts for the text image associated with the process, for all processes that share that text. It must be run while the program is still active, since the data is stored with the running program. To enable `tprof` profiling for a given process,

```
echo profile > /proc/pid/ctl
```

and then, after the program has run for a while, execute

```
tprof pid
```

Since the data collected for `tprof` is based on interrupt–time sampling of the program counter, `tprof` has no `-d` or `-r` options.

`kprof` is similar to `prof`, but presents the data accumulated by the kernel profiling device, `kprof(3)`. The symbol table file, that of the operating system kernel, and the data file, typically `/dev/kpdata`, must be provided. `kprof` has no options and cannot present dynamic data.

**SOURCE**

`/sys/src/cmd/prof.c`
/sys/src/cmd/kprof.c

SEE ALSO
2l(1), exec(2), kprof(3)
NAME
   proof – troff output interpreter

SYNOPSIS
   proof [-mmag] [-/nview] [-F dir] [-d] [file]

DESCRIPTION
   Proof reads troff(1) intermediate language from file or standard input and simulates the resulting
   pages on the screen.

   After a page of text is displayed, proof pauses for a command from the keyboard. The typed com-
   mands are:
   newline    Go on to next page of text.
   –           Go back to the previous page.
   q           Quit.
   pn          Print page n. An out-of-bounds page number means the end nearer to that number; a
               missing number means the current page; a signed number means an offset to the current
               page.
   n           Same as pn.
   c           Clear the screen, then wait for another command.
   mmag        Change the magnification at which the output is printed. Normally it is printed with mag-
               nification .9; mag=.5 shrinks it to half size; mag=2 doubles the size.
   xval        Move everything val screen pixels to the right (left, if val is negative).
   yval        Move everything val screen pixels down (up, if val is negative).
   /nview      Split the window into nview pieces. The current page goes into the rightmost, bottom-
               most piece, and previous pages are shown in the other pieces.
   -F dir      Use dir for fonts instead of /lib/font/bit.
   d           Toggle the debug flag.

   These commands are also available, under slightly different form, from a menu on button 3. The
   pan menu item allows arbitrary positioning of the page: after selecting pan, press the mouse but-
   ton again and hold it down while moving the page to the desired location. The page will be redis-
   played in its entirety when the button is released. Mouse button 1 also pans, without the need for
   selecting from a menu.

   The m, x, y, F, /, and d commands are also available as command line options.

FILES
   /lib/font/bit/*  fonts
   /lib/font/bit/MAP how to convert troff output fonts and character names into screen
                   fonts and character numbers

SOURCE
   /sys/src/cmd/proof

SEE ALSO
   lp(1), gs(1), page(1)
NAME
ps, psu - process status

SYNOPSIS
ps [−apr]
psu [−apr][user]

DESCRIPTION
Ps prints information about processes. Psu prints only information about processes started by user (default $USER).

For each process reported, the user, process id, user time, system time, size, state, and command name are printed. State is one of the following:

- Moribund: Process has exited and is about to have its resources reclaimed.
- Ready: On the queue of processes ready to be run.
- Scheduling: About to be run.
- Running: Running.
- Queueing: Waiting on a queue for a resource.
- Wakeme: Waiting for I/O or some other kernel event to wake it up.
- Broken: Dead of unnatural causes; lingering so that it can be examined.
- Stopped: Stopped.
- Stopwait: Waiting for another process to stop.
- Fault: Servicing a page fault.
- Idle: Waiting for something to do (kernel processes only).
- New: Being created.
- Pageout: Paging out some other process.
- Syscall: Performing the named system call.
- no resource: Waiting for more of a critical resource.

The −r flag causes ps to print, before the user time, the elapsed real time for the process.
The −p flag causes ps to print, after the system time, the baseline and current priorities of each process.
The −a flag causes ps to print the arguments for the process. Newlines in arguments will be translated to spaces for display.

FILES
/proc/*/status

SOURCE
/sys/src/cmd/ps.c
/rc/bin/psu

SEE ALSO
acid(1), db(1), kill(1), ns(1), proc(3)
NAME
ps2pdf, pdf2ps – convert between PostScript and PDF

SYNOPSIS
ps2pdf [ gs−options ] [ input−file [ output−file ]]
pdf2ps [ gs−options ] [ input−file [ output−file ]]

DESCRIPTION
Ps2pdf and pdf2ps convert from PostScript to PDF and back by invoking gs(1). If output−file is not specified, they write to standard output. If neither input−file nor output−file is not specified, they read from standard input and write to standard output.

The gs−options are passed to Ghostscript unaltered. The most useful option to ps2pdf is −dCompatibilityLevel=level, which sets the version of PDF to be written. The default is 1.2; 1.3 and 1.4 are also possible. Similarly, the most useful option to pdf2ps is −dLanguageLevel=level, which sets the version of PostScript to be written. The default is 2; 1 and 3 are also possible.

Ps2pdf produces output competitive with Adobe Distiller in most cases, and it accepts all the embedded PDF−generation hints that Adobe Distiller does.

Pdf2ps produces a PostScript file containing one large bitmap per page. For a more direct and smaller translation, use Adobe Acrobat’s −toPostScript command−line option.

SOURCE
/rc/bin/ps2pdf
/rc/bin/pdf2ps

SEE ALSO
gs(1)

BUGS
Gs’s pdfwrite sometimes emits bad PDF at the default level 1.2. Adding ’−dCompatibilityLevel=1.4’ should cure it.
NAME
pump - copy asynchronously via a large circular buffer

SYNOPSIS
pump [ -b iando ] [ -d sleepms ] [ -f ofile ] [ -i ireadsize ] [ -k KB-buf ] [ -o owritesize ] [ -s start-KB ] [ file ... ]

DESCRIPTION
Pump copies files (or standard input if none) to standard output by using two processes, one reading and one writing, sharing a large circular buffer, thus permitting the reading process to get ahead of the writing process if the output device is slow (e.g., an optical disc). This in turn can keep the output device busy. The pipeline dd | dd can approximate this, but pipe buffering is limited to 64K bytes, which is fairly modest.

Options are:
- `b` sets the size of read and write operations to iando bytes. The default size is 8 kilobytes.
- `d` causes the output process to sleep for sleepms milliseconds initially, giving the reading process time to accumulate data in the buffer.
- `f` writes ofile rather than standard output
- `i` sets the size of read operations to ireadsize bytes.
- `k` allocates a circular buffer of KB-buf kilobytes rather than the default 5000 kilobytes.
- `o` sets the size of write operations to owritesize bytes.
- `s` prevents output until start-KB kilobytes have been read.

EXAMPLES
Append a venti(8) arena to a DVD or BD quickly.
```
cdfs
  venti/rdarena arena0 arena.3 |
  pump -b 65536 -k 51200 >/mnt/cd/wd/arena.3
```

SOURCE
/sys/src/cmd/pump.c

SEE ALSO
`cp(1), dd(1), ecp(1), cdfs(4)`

BUGS
Pump processes spin while waiting for the circular buffer to fill or drain.
NAME
   pwd, pbd – working directory

SYNOPSIS
   pwd
   pbd

DESCRIPTION
   Pwd prints the path name of the working (current) directory. Pwd is guaranteed to return the same path that was used to enter the directory. If, however, the name space has changed, or directory names have been changed, this path name may no longer be valid. (See fd2path(2) for a description of pwd’s mechanism.)

   Pbd prints the base name of the working (current) directory. It prints no final newline and is intended for applications such as constructing shell prompts.

SOURCE
   /sys/src/cmd/pwd.c
   /sys/src/cmd/pbd.c

SEE ALSO
   cd in rc(1), bind(1), intro(2), getwd(2), fd2path(2)
NAME
ratrace – trace process system calls

SYNOPSIS
ratrace [ pid ] | [ -c command ]

DESCRIPTION
Ratrace shows the system calls executed by a process, either the one with pid or a fresh invocation of command.

Trace output is determined by the kernel, not ratrace. Certain fixed rules apply. The first four fields of the output are pid, text name, system call name, and the PC of the user program. Data is always printed as pointer/"string", where the string is the first 32 bytes of the data, with . replacing non-printing ASCII characters (printing characters are those between ASCII space (SP) and delete (DEL), exclusive). Return values follow an =, and include the integer return value, the errstr (with "" if there is no errstr), and the start and stop times for the system call in nanoseconds. The times are exclusive of the overhead for tracing.

FILES
/proc/pid/syscalltrace
/proc/pid/ctl

SOURCE
/sys/src/cmd/ratrace.c

SEE ALSO
acid(1), db(1), proc(3)

BUGS
The printing of the data is too limited in length; printing . instead of something more sensible is limiting.
NAME
rc, cd, eval, exec, exit, flag, rfork, shift, wait, whatis, .. ~ - command language

SYNOPSIS
rc [-srdIlxepV][ -c command][ -m initial][ file [ arg ...]]

DESCRIPTION
Rc is the Plan 9 shell. It executes command lines read from a terminal or a file or, with the -c flag, from rc’s argument list.

Command Lines
A command line is a sequence of commands, separated by ampersands or semicolons (& or ;), terminated by a newline. The commands are executed in sequence from left to right. Rc does not wait for a command followed by & to finish executing before starting the following command. Whenever a command followed by & is executed, its process id is assigned to the rc variable $apid. Whenever a command not followed by & exits or is terminated, the rc variable $status gets the process’s wait message (see wait(2)); it will be the null string if the command was successful.

A long command line may be continued on subsequent lines by typing a backslash (\) followed by a newline. This sequence is treated as though it were a blank. Backslash is not otherwise a special character.

A number–sign (#) and any following characters up to (but not including) the next newline are ignored, except in quotation marks.

Simple Commands
A simple command is a sequence of arguments interspersed with I/O redirections. If the first argument is the name of an rc function or of one of rc’s built-in commands, it is executed by rc. Otherwise if the name starts with a slash (/), it must be the path name of the program to be executed. Names containing no initial slash are searched for in a list of directory names stored in $path. The first executable file of the given name found in a directory in $path is the program to be executed. To be executable, the user must have execute permission (see stat(2)) and the file must be either an executable binary for the current machine’s CPU type, or a shell script. Shell scripts begin with a line containing the full path name of a shell (usually /bin/rc), prefixed by #!

The first word of a simple command cannot be a keyword unless it is quoted or otherwise disguised. The keywords are
   for in while if not switch fn ~ !@

Arguments and Variables
A number of constructions may be used where rc’s syntax requires an argument to appear. In many cases a construction’s value will be a list of arguments rather than a single string.

The simplest kind of argument is the unquoted word: a sequence of one or more characters none of which is a blank, tab, newline, or any of the following:
   # ; & | ^ $ = ' ' { } ( ) < >

An unquoted word that contains any of the characters * ? [ ] is a pattern for matching against file names. The character * matches any sequence of characters, ? matches any single character, and [class] matches any character in the class. If the first character of class is ~, the class is complemented. The class may also contain pairs of characters separated by ~, standing for all characters lexically between the two. The character / must appear explicitly in a pattern, as must the first character of the path name components . and .. A pattern is replaced by a list of arguments, one for each path name matched, except that a pattern matching no names is not replaced by the empty list, but rather stands for itself. Pattern matching is done after all other operations. Thus,
   x=/tmp echo $x/*
matches /tmp/* and then prefixing /tmp.

A quoted word is a sequence of characters surrounded by single quotes (‘). A single quote is represented in a quoted word by a pair of quotes (’ ’).

Each of the following is an argument.
( arguments)

The value of a sequence of arguments enclosed in parentheses is a list comprising the
members of each element of the sequence. Argument lists have no recursive structure, although their syntax may suggest it. The following are entirely equivalent:

```
  echo hi there everybody
  ((echo) (hi there) everybody)
```

$argument
The argument after the $ is the name of a variable whose value is substituted. Multiple levels of indirection are possible, but of questionable utility. Variable values are lists of strings. If argument is a number n, the value is the nth element of $^n$, unless $^n$ doesn’t have n elements, in which case the value is empty. If argument is followed by a parenthesized list of subscripts, the value substituted is a list composed of the requested elements (origin 1). The parenthesis must follow the variable name with no spaces. Subscripts can also take the form m−n or m− to indicate a sequence of elements. Assignments to variables are described below.

$#argument
The value is the number of elements in the named variable. A variable never assigned a value has zero elements.

$"argument
The value is a single string containing the components of the named variable separated by spaces. A variable with zero elements yields the empty string.

`{command}
rc executes the command and reads its standard output, splitting it into a list of arguments, using characters in $ifs as separators. If $ifs is not otherwise set, its value is ' \t\n'.

`<{command}
`>{command}
The command is executed asynchronously with its standard output or standard input connected to a pipe. The value of the argument is the name of a file referring to the other end of the pipe. This allows the construction of non-linear pipelines. For example, the following runs two commands old and new and uses cmp to compare their outputs:

```
cmp <{old} <{new}
```

$argument^argument
The ^ operator concatenates its two operands. If the two operands have the same number of components, they are concatenated pairwise. If not, then one operand must have one component, and the other must be non-empty, and concatenation is distributive.

Free Carets
In most circumstances, rc will insert the ^ operator automatically between words that are not separated by white space. Whenever one of $ ^' follows a quoted or unquoted word or an unquoted word follows a quoted word with no intervening blanks or tabs, a ^ is inserted between the two. If an unquoted word immediately follows a $ and contains a character other than an alphanumeric, underscore, or $, a ^ is inserted before the first such character. Thus

```
cc -$flags $stem.c
```

is equivalent to

```
cc -$^flags $stem^.c
```

I/O Redirections
The sequence >file redirects the standard output file (file descriptor 1, normally the terminal) to the named file; >>file appends standard output to the file. The standard input file (file descriptor 0, also normally the terminal) may be redirected from a file by the sequence <file, or from an inline ‘here document’ by the sequence <<<eof−marker. The contents of a here document are lines of text taken from the command input stream up to a line containing nothing but the eof−marker, which may be either a quoted or unquoted word. If eof−marker is unquoted, variable names of the form $word have their values substituted from rc’s environment. If $word is followed by a caret (^), the caret is deleted. If eof−marker is quoted, no substitution occurs. The standard input file may also be redirected from a file by the sequence <>file, which opens file exactly once, for reading and writing.

Redirections may be applied to a file−descriptor other than standard input or output by qualifying the redirection operator with a number in square brackets. For example, the diagnostic output
(file descriptor 2) may be redirected by writing
\texttt{cc junk.c >>2} junk.

A file descriptor may be redirected to an already open descriptor by writing
\texttt{>>} [fd0=fd1], or \texttt{<<} [fd0=fd1]. fd1 is a previously opened file descriptor and fd0 becomes a
new copy (in the sense of \texttt{dup}) of it. A file descriptor may be closed by writing \texttt{>>} [fd0=] or
\texttt{<<} [fd0=].

Redirections are executed from left to right. Therefore, \texttt{cc junk.c >>4} /dev/null \texttt{>>2}=
\texttt{1} and \texttt{cc junk.c >>2} \texttt{=} \texttt{1} \texttt{>>} /dev/null have different effects: the first puts
standard output in \texttt{/dev/null} and then puts diagnostic output in the same place, where the second
directs diagnostic output to the terminal and sends standard output to \texttt{/dev/null}.

\texttt{newconn} \texttt{>>} /net/tcp/clone \texttt{>>} \texttt{0} opens \texttt{/net/tcp/clone} exactly once for reading
and writing and puts it on standard input and output. \texttt{lpd} \texttt{<<} \texttt{3} \texttt{/net/tcp/42/data} opens
\texttt{/net/tcp/42/data} exactly once for reading and writing and puts it on file descriptor 3.

\textbf{Compound Commands}

A pair of commands separated by a pipe operator (\texttt{\textbars}) is a command. The standard output of the
left command is sent through a pipe to the standard input of the right command. The pipe operator
may be decorated to use different file descriptors. \texttt{\textbars} \texttt{[fd]} connects the output end of the pipe
to file descriptor \texttt{fd} rather than 1. \texttt{\textbars} \texttt{[fd0=fd1]} connects output to \texttt{fd1} of the left command and
input to \texttt{fd0} of the right command.

A pair of commands separated by \&\& or \texttt{||} is a command. In either case, the left command is
executed and its exit status examined. If the operator is \&\& the right command is executed if the
left command’s status is null. \texttt{||} causes the right command to be executed if the left command’s
status is non–null.

The exit status of a command may be inverted (non–null is changed to null, null is changed to
non–null) by preceding it with a \texttt{!}.

The \texttt{!} operator has highest precedence, and is left–associative (i.e. binds tighter to the left than
the right). \texttt{!} has intermediate precedence, and \&\& and \texttt{||} have the lowest precedence.

The unary \texttt{@} operator, with precedence equal to \texttt{!}, causes its operand to be executed in a subshell.

Each of the following is a command.

\begin{itemize}
  \item \texttt{if ( list ) command}
    A list is a sequence of commands, separated by \&, \texttt{;}, or newline. It is executed and if its
    exit status is null, the \texttt{command} is executed.
  \item \texttt{if not command}
    The immediately preceding command must have been \texttt{if(list) command}. If its condition
    was non–zero, the \texttt{command} is executed.
  \item \texttt{for(name in arguments) command}
  \item \texttt{for(name) command}
    The \texttt{command} is executed once for each \texttt{argument} with that argument assigned to \texttt{name}.
    If the argument list is omitted, \texttt{$*\texttt{}} is used.
  \item \texttt{while(list) command}
    The \texttt{list} is executed repeatedly until its exit status is non–null. Each time it returns
    null status, the \texttt{command} is executed. An empty \texttt{list} is taken to give null status.
  \item \texttt{switch(argument){list}}
    The \texttt{list} is searched for simple commands beginning with the word \texttt{case}. (The search
    is only at the ‘top level’ of the \texttt{list}. That is, \texttt{cases} in nested constructs are not found.)
    \texttt{Argument} is matched against each word following \texttt{case} using the pattern–matching
    algorithm described above, except that / and the first characters of . and \ldots need not be
    matched explicitly. When a match is found, commands in the list are executed up to the
    next following \texttt{case} command (at the top level) or the closing brace.

\end{itemize}

\texttt{\{} \texttt{list}\texttt{\}}

Braces serve to alter the grouping of commands implied by operator priorities. The \texttt{body} is
a sequence of commands separated by \&, \texttt{;}, or newline.

\begin{itemize}
  \item \texttt{fn name{list}}
    \texttt{fn name}
    The first form defines a function with the given \texttt{name}. Subsequently, whenever a command
    whose first argument is \texttt{name} is encountered, the current value of the remainder of the
    command’s argument list will be assigned to \texttt{$*\texttt{}}\texttt{,} after saving its current value, and \texttt{rc}
    will
execute the \textit{list}. The second form removes \textit{name}'s function definition.

\begin{verbatim}
\texttt{fn \{note\}}
\texttt{fn \{note\}}
\end{verbatim}

A function with a special name will be called when \texttt{rc} receives a corresponding note; see \texttt{notify(2)}. The valid note names (and corresponding notes) are \texttt{sigtyp (hangup)}, \texttt{sigint (interrupt)}, \texttt{sigalarm (alarm)}, and \texttt{sigfpe (floating point trap)}. By default \texttt{rc} exits on receiving any signal, except when run interactively, in which case interrupts and quits normally cause \texttt{rc} to stop whatever it's doing and start reading a new command. The second form causes \texttt{rc} to handle a signal in the default manner. \texttt{rc} recognizes an artificial note, \texttt{sigexit}, which occurs when \texttt{rc} is about to finish executing.

\texttt{name=argument command}

Any command may be preceded by a sequence of assignments interspersed with redirections. The assignments remain in effect until the end of the command, unless the command is empty (i.e. the assignments stand alone), in which case they are effective until rescinded by later assignments.

\textbf{Built-in Commands}

These commands are executed internally by \texttt{rc}, usually because their execution changes or depends on \texttt{rc}'s internal state.

\begin{verbatim}
\texttt{. \{} \texttt{file \}}
\end{verbatim}

Execute commands from \texttt{file}. \texttt{*} is set for the duration to the remainder of the argument list following \texttt{file}. \texttt{File} is searched for using \texttt{path}.

\begin{verbatim}
\texttt{builtin \{} \texttt{command \}}
\end{verbatim}

Execute \texttt{command} as usual except that any function named \texttt{command} is ignored in favor of the built-in meaning.

\begin{verbatim}
\texttt{cd \{dir\}}
\end{verbatim}

Change the current directory to \texttt{dir}. The default argument is \texttt{$home}. \texttt{dir} is searched for in each of the directories mentioned in \texttt{cdpath}.

\begin{verbatim}
\texttt{eval \{arg \}}
\end{verbatim}

The arguments are concatenated separated by spaces into a single string, read as input to \texttt{rc}, and executed.

\begin{verbatim}
\texttt{exec \{command \}}
\end{verbatim}

This instance of \texttt{rc} replaces itself with the given (non-built-in) \texttt{command}.

\begin{verbatim}
\texttt{flag \{f \}}
\end{verbatim}

Either set (+), clear (−), or test (neither + nor −) the flag \texttt{f}, where \texttt{f} is a single character, one of the command line flags (see Invocation, below).

\begin{verbatim}
\texttt{exit \{status\}}
\end{verbatim}

Exit with the given exit status. If none is given, the current value of \texttt{$status} is used.

\begin{verbatim}
\texttt{rfork \{n\NeE\}}
\end{verbatim}

Become a new process group using \texttt{rfork(flags)} where \texttt{flags} is composed of the bitwise OR of the \texttt{rfork} flags specified by the option letters (see \texttt{fork(2)}). If no \texttt{flags} are given, they default to ens. The flags and their meanings are: n is \texttt{RFNAMEG}; N is \texttt{RFCNAMEG}; e is \texttt{RFENVG}; E is \texttt{RFCENVG}; s is \texttt{RFNOTEQG}; f is \texttt{RFFDG}; F is \texttt{RFCFEDG}; and m is \texttt{RFNOMNT}.

\begin{verbatim}
\texttt{shift \{n\}}
\end{verbatim}

Delete the first \texttt{n} (default 1) elements of \texttt{*}.

\begin{verbatim}
\texttt{wait \{pid\}}
\end{verbatim}

Wait for the process with the given \texttt{pid} to exit. If no \texttt{pid} is given, all outstanding processes are waited for.

\begin{verbatim}
\texttt{what\{is \}}
\end{verbatim}

Print the value of each \texttt{name} in a form suitable for input to \texttt{rc}. The output is an assignment to any variable, the definition of any function, a call to \texttt{builtin} for any built-in command, or the completed pathname of any executable file.

\begin{verbatim}
\texttt{\sim \sim \sim \sim subject \sim \sim \sim \sim pattern \}}
\end{verbatim}

The \texttt{subject} is matched against each \texttt{pattern} in sequence. If it matches any pattern, \texttt{$status} is set to zero. Otherwise, \texttt{$status} is set to one. Patterns are the same as for file name matching, except that / and the first character of .. and .. need not be matched explicitly. The \texttt{patterns} are not subjected to file name matching before the \texttt{\sim} command is executed, so they need not be enclosed in quotation marks.
Environment

The environment is a list of strings made available to executing binaries by the env device (see env(3)). Rc creates an environment entry for each variable whose value is non-empty, and for each function. The string for a variable entry has the variable's name followed by = and its value. If the value has more than one component, these are separated by control-a ('\01') characters. The string for a function is just the rc input that defines the function. The name of a function in the environment is the function name preceded by fn#.

When rc starts executing it reads variable and function definitions from its environment.

Special Variables

The following variables are set or used by rc.

$* Set to rc's argument list during initialization. Whenever a . command or a function is executed, the current value is saved and $* receives the new argument list. The saved value is restored on completion of the . or function.

$apid Whenever a process is started asynchronously with & $apid is set to its process id.

$ifs The input field separators used in backquote substitutions. If $ifs is not set in rc's environment, it is initialized to blank, tab, and newline.

$path The search path used to find commands and input files for the . command. If not set in the environment, it is initialized by path=./bin. Its use is discouraged; instead use bind(1) to build a /bin containing what's needed.

$pid Set during initialization to rc's process id.

$prompt When rc is run interactively, the first component of $prompt is printed before reading each command. The second component is printed whenever a newline is typed and more lines are required to complete the command. If not set in the environment, it is initialized by prompt="('% ' ' ')".

$status Set to the wait message of the last-executed program. (unless started with &). ! and ~ also change $status. Its value is used to control execution in &&, ||, if and while commands. When rc exits at end-of-file of its input or on executing an exit command with no argument, $status is its exit status.

Invocation

If rc is started with no arguments it reads commands from standard input. Otherwise its first non-flag argument is the name of a file from which to read commands (but see -c below). Subsequent arguments become the initial value of $*. Rc accepts the following command-line flags.

-c string Commands are read from string.

-s Print out exit status after any command where the status is non-null.

-e Exit if $status is non-null after executing a simple command.

-i If -i is present, or rc is given no arguments and its standard input is a terminal, it runs interactively. Commands are prompted for using $prompt.

-I Makes sure rc is not run interactively.

-l If -l is given or the first character of argument zero is -, rc reads commands from $home/lib/profile, if it exists, before reading its normal input.

-m Read commands to initialize rc from initial instead of from /rc/lib/rcmain.

-p A no-op.

-d A no-op.

-v Echo input on file descriptor 2 as it is read.

-x Print each simple command before executing it.

-r Print debugging information (internal form of commands as they are executed).

SOURCE
/sys/src/cmd/rc

SEE ALSO
Tom Duff, “Rc – The Plan 9 Shell”.

BUGS

There should be a way to match patterns against whole lists rather than just single strings.

Using ~ to check the value of $status changes $status.

Functions containing here documents don’t work.
Free carets don’t get inserted next to keywords.
NAME
changes, pull, push, scan - client–server replica management

SYNOPSIS
replica/pull [-nv][-c name]... [-s name]... name [path]
replica/push [-nv] name [path]
replica/changes name [path]
replica/scan name [path]

DESCRIPTION
These shell scripts provide a simple log–based client–server replica management. The server
keeps a log of changes made to its file system, and clients synchronize by reading the log and
applying these changes locally.

These scripts are a polished interface to the low–level tools described in replica(8). See replica(8)
for details on the inner workings of replica management. These tools were written primarily as the
fourth edition Plan 9 distribution mechanism, but they have wider applicability. For example, they
could be used to synchronize one’s home directory between a laptop and a central file server.

Replicas are described by configuration files. The name in all the replica commands is a configura-
tion file. Paths that do not begin with /, ./, or ../ are assumed to be relative to $home/lib/replica. Configuration files are described below.

Replica/scan is the only one of these programs that does not need to be run on the client. It scans
the server file system for changes and appends entries for those changes into the server log. Typi-
cally it is run on a machine with a fast network connection to the server file system.

Replica/pull copies changes from the server to the client, while replica/push copies changes from
the client to the server. (Both run on the client.) If a list of paths is given, only changes to those
paths or their children are copied. The -v flag causes pull or push to print a summary of what it is
doing. Each status line is of the form

verb path serverpath mode uid gid mtime length

Verb describes the event: addition of a file (a), deletion of a file (d), a change to a file’s contents
(c), or a change to a file’s metadata (m). Path is the file path on the client; serverpath is the file
path on the server. Mode, uid, gid, and mtime are the file’s metadata as in the Dg structure (see
stat(5)). For deletion events, the metadata is that of the deleted file. For other events, the meta-
data is that after the event. The -n flag causes pull or push to print the summary but not actually
carry out the actions.

Push and pull are careful to notice simultaneous changes to a file or its metadata on both client
and server. Such simultaneous changes are called conflicts. Here, simultaneous does not mean at
the same instant but merely that both changes were carried out without knowledge of the other.
For example, if a client and server both make changes to a file without an intervening push or pull,
the next push or pull will report an update/update conflict. If a conflict is detected, both files are
left the same. The -c flag to pull specifies that conflicts for paths beginning with name should be
resolved using the client’s copy, while -s specifies the server’s copy. The -c and -s options may
be repeated.

Replica/changes prints a list of local changes made on the client that have not yet been pushed to
the server. It is like push with the -n flag, except that it does not check for conflicts and thus
does not require the server to be available.

The replica configuration file is an rc(1) script that must define the following functions and vari-
bles:

servermount
A function that mounts the server; run on both client and server.

serverupdate
A function that rescans the server for changes. Typically this command dials a CPU server
known to be close to the file server and runs replica/scan on that well–connected machine.

serverroot
The path to the root of the replicated file system on the server, as it will be in the name
space after running servermount.

serverlog
The path to the server's change log, after running servermount.

serverproto
The path to the proto file describing the server's files, after running servermount. Only used by scan.

serverdb
The path to the server's file database, after running servermount. Only used by scan.

clientmount
A function to mount the client file system; run only on the client.

clientroot
The path to the root of the replicated file system on the client, after running clientmount.

clientlog
The path to the client's copy of the server log file. The client log is maintained by pull.

clientproto
The path to the proto file describing the client's files. Only used by changes. Often just a copy of $serverproto.

clientdb
The path to the client's file database, after running clientmount.

clientexclude
A (potentially empty) list of paths to exclude from synchronization. A typical use of this is to exclude the client database and log files. These paths are relative to the root of the replicated file system.

As an example, the Plan 9 distribution replica configuration looks like:

```
fn servermount { 9fs sources; bind /n/sources/plan9 /n/dist }
fn serverupdate { status='' }
serverroot=/n/dist
s=/n/dist/dist/replica
serverlog=$s/plan9.log
serverproto=$s/plan9.proto

fn clientmount { 9fs kfs }
clientroot=/n/kfs
kfs=/n/kfs/dist/replica
clientlog=$c/client/plan9.log
clientproto=$c/plan9.proto
clientdb=$c/client/plan9.db
clientexclude=(dist/replica/client)
```

(Since the Plan 9 developers run scan manually to update the log, the clients need not do anything to rescan the file system. Thus serverupdate simply returns successfully.)

The fourth edition Plan 9 distribution uses these tools to synchronize installations with the central server at Bell Labs. The replica configuration files and metadata are kept in /dist/replica. To update your system, make sure you are connected to the internet and run

```
replica/pull /dist/replica/network
```

If conflicts are reported (say you have made local changes to /rc/bin/cpurc and /rc/bin/termrc, but only want to keep the cpurc changes), use

```
replica/pull -c rc/bin/cpurc -s rc/bin/termrc /dist/replica/network
```

to instruct pull to ignore the server's change to cpurc.

The script /usr/glenda/bin/rc/pull runs pull with the -v flag and with /dist/replica/network inserted at the right point on the command line. Logged in as glenda, one can repeat the above example with:

```
pull -c rc/bin/cpurc -s rc/bin/termrc
```
To see a list of changes made to the local file system since installation, run

`replica/changes /dist/replica/network`

(Although the script is called `network`, since `changes` is a local-only operation, the network need not be configured.)

**SOURCE**

```
/rc/bin/replica
```

**SEE ALSO**

`replica(8)`
NAME
resample - resample a picture

SYNOPSIS
resample [-x size] [-y size] [file]

DESCRIPTION
Resample resamples its input image (default standard input) to a new size. The image is decimated or interpolated using a Kaiser window.

The size of the resampled image can be specified with the -x and -y options. An unadorned value sets the number of pixels of that dimension; a suffixed percent sign specifies a percentage. If only one of -x or -y is given, the other dimension is scaled to preserve the aspect ratio of the original image. Thus, -x50% will reduce the image to half its original dimension in both x and y.

The input should be a Plan 9 image as described in image(6), and the output will be a compressed 24-bit r8g8b8 image. To uncompress the image or change the pixel format, use iconv (see crop(1)).

SOURCE
/sys/src/cmd/resample.c

SEE ALSO
crop(1), image(6)

BUGS
Faster algorithms exist, but this implementation produces correct pictures.
NAME
rio, label, window, wloc - window system

SYNOPSIS
rio [-i 'cmd'][-k 'kbdcmd'][-s][-f font]
label name
window [-m][-x minx miny maxx maxy][-dx n][-dy n][-minx n][-miny n][-maxx n][-maxy n][-cd dir][-hide][-scroll][-noscroll][cmd arg ...]
wloc

DESCRIPTION
RIO manages asynchronous layers of text, or windows, on a raster display. It also serves a variety of files for communicating with and controlling windows; these are discussed in section rio(4).

Commands
The rio command starts a new instance of the window system. Its -i option names a startup script, which typically contains several window commands generated by wloc. The -k option causes rio to run the command kbdcmd at startup and allow it to provide characters as keyboard input; the keyboard program described in bitsyload(1) is the usual choice.

The -s option initializes windows so that text scrolls; the default is not to scroll. The font argument names a font used to display text, both in rio's menus and as a default for any programs running in its windows; it also establishes the environment variable $font. If -f is not given, rio uses the imported value of $font if set; otherwise it imports the default font from the underlying graphics server, usually the terminal's operating system.

The label command changes a window's identifying name.

The window command creates a window. By default, it creates a shell window and sizes and places it automatically. The geometry arguments control the size (dx, dy) and placement (minx, miny, maxx, maxy); the units are pixels with the upper left corner of the screen at (0, 0). The hide option causes the window to be created off-screen. The scroll and noscroll options set the scroll mode. The cd option sets the working directory. The optional command and arguments define which program to run in the window.

By default, window uses /dev/wctl (see rio(4)) to create the window and run the command. Therefore, the window and command will be created by rio and run in a new file name space, just as if the window had been created using the interactive menu. However, the -m option uses the file server properties of rio to mount (see bind(1)) the new window's name space within the name space of the program calling window. This means, for example, that running window in a CPU window will create another window whose command runs on the terminal, where rio is running; while window -m will create another window whose command runs on the CPU server.

The wloc command prints the coordinates and label of each window in its instance of rio and is used to construct arguments for window.

Window control
Each window behaves as a separate terminal with at least one process associated with it. When a window is created, a new process (usually a shell; see rc(1)) is established and bound to the window as a new process group. Initially, each window acts as a simple terminal that displays character text; the standard input and output of its processes are attached to /dev/cons. Other special files, accessible to the processes running in a window, may be used to make the window a more general display. Some of these are mentioned here; the complete set is discussed in rio(4).

One window is current, and is indicated with a dark border and text; characters typed on the keyboard are available in the /dev/cons file of the process in the current window. Characters written on /dev/cons appear asynchronously in the associated window whether or not the window is current.

Windows are created, deleted and rearranged using the mouse. Clicking (pressing and releasing) mouse button 1 in a non–current window makes that window current and brings it in front of any windows that happen to be overlapping it. When the mouse cursor points to the background area or is in a window that has not claimed the mouse for its own use, pressing mouse button 3
activates a menu of window operations provided by _rio_. Releasing button 3 then selects an operation. At this point, a gunsight or cross cursor indicates that an operation is pending. The button 3 menu operations are:

- **New** Create a window. Press button 3 where one corner of the new rectangle should appear (cross cursor), and move the mouse, while holding down button 3, to the diagonally opposite corner. Releasing button 3 creates the window, and makes it current. Very small windows may not be created.

- **Resize** Change the size and location of a window. First click button 3 in the window to be changed (gunsight cursor). Then sweep out a window as for the New operation. The window is made current.

- **Move** Move a window to another location. After pressing and holding button 3 over the window to be moved (gunsight cursor), indicate the new position by dragging the rectangle to the new location. The window is made current. Windows may be moved partially off-screen.

- **Delete** Delete a window. Click in the window to be deleted (gunsight cursor). Deleting a window causes a hangup note to be sent to all processes in the window’s process group (see _notify(2)_).

- **Hide** Hide a window. Click in the window to be hidden (gunsight cursor); it will be moved off-screen. Each hidden window is given a menu entry in the button 3 menu according to the value of the file `/dev/label`, which _rio_ maintains (see _rio(4)_).

- **label** Restore a hidden window.

Windows may also be arranged by dragging their borders. Pressing button 1 or 2 over a window’s border allows one to move the corresponding edge or corner, while button 3 moves the whole window.

**Text windows**

Characters typed on the keyboard or written to `/dev/cons` collect in the window to form a long, continuous document.

There is always some **selected text**, a contiguous string marked on the screen by reversing its color. If the selected text is a null string, it is indicated by a hairline cursor between two characters. The selected text may be edited by mousing and typing. Text is selected by pointing and clicking button 1 to make a null-string selection, or by pointing, then sweeping with button 1 pressed. Text may also be selected by double-clicking: just inside a matched delimiter-pair with one of `{ [(<<' ''' on the left and '''' on the right, it selects all text within the pair; at the beginning or end of a line, it selects the line; within or at the edge of an alphanumerical word, it selects the word.

Characters typed on the keyboard replace the selected text; if this text is not empty, it is placed in a **snarf buffer** common to all windows but distinct from that of _sam(1)_.

Programs access the text in the window at a single point maintained automatically by _rio_. The **output point** is the location in the text where the next character written by a program to `/dev/cons` will appear; afterwards, the output point is the null string beyond the new character. The output point is also the location in the text of the next character that will be read (directly from the text in the window, not from an intervening buffer) by a program from `/dev/cons`. When such a read will occur is, however, under control of _rio_ and the user.

In general there is text in the window after the output point, usually placed there by typing but occasionally by the editing operations described below. A pending read of `/dev/cons` will block until the text after the output point contains a newline, whereupon the read may acquire the text, up to and including the newline. After the read, as described above, the output point will be at the beginning of the next line of text. In normal circumstances, therefore, typed text is delivered to programs a line at a time. Changes made by typing or editing before the text is read will not be seen by the program reading it. If the program in the window does not read the terminal, for example if it is a long-running computation, there may accumulate multiple lines of text after the output point; changes made to all this text will be seen when the text is eventually read. This means, for example, that one may edit out newlines in unread text to forestall the associated text being read when the program finishes computing. This behavior is very different from most systems.
Even when there are newlines in the output text, r.io will not honor reads if the window is in hold mode, which is indicated by a white cursor and blue text and border. The ESC character toggles hold mode. Some programs, such as mail(1), automatically turn on hold mode to simplify the editing of multi-line text; type ESC when done to allow mail to read the text.

An EOT character (control-D) behaves exactly like newline except that it is not delivered to a program when read. Thus on an empty line an EOT serves to deliver an end-of-file indication: the read will return zero characters. Like newlines, unread EOTs may be successfully edited out of the text. The BS character (control-H) erases the character before the selected text. The ETB character (control-W) erases any non-alphanumeric characters, then the alphanumeric word just before the selected text. ‘Alphanumeric’ here means non-blanks and non-punctuation. The NAK character (control-U) erases the text after the output point, and not yet read by a program, but not more than one line. All these characters are typed on the keyboard and hence replace the selected text; for example, typing a BS with a word selected places the word in the snarf buffer, removes it from the screen, and erases the character before the word.

An ACK character (control-F) or Insert character triggers file name completion for the preceding string (see complete(2)).

Typing a left or right arrow moves the cursor one character in that direction. Typing an SOH character (control-A) moves the cursor to the beginning of the current line; an ENQ character (control-E) moves to the end.

Text may be moved vertically within the window. A scroll bar on the left of the window shows in its clear portion what fragment of the total output text is visible on the screen, and in its gray part what is above or below view; it measures characters, not lines. Mousing inside the scroll bar moves text: clicking button 1 with the mouse pointing inside the scroll bar brings the line at the top of the window to the cursor’s vertical location; button 3 takes the line at the cursor to the top of the window; button 2, treating the scroll bar as a ruler, jumps to the indicated portion of the stored text. Holding a button pressed in the scroll bar will cause the text to scroll continuously until the button is released. Also, a page down or down–arrow scrolls forward half a window, and page up or up–arrow scrolls back. Typing the home key scrolls to the top of the window; typing the end key scrolls to the bottom.

The DEL character sends an interrupt note to all processes in the window’s process group. Unlike the other characters, the DEL, VIEW, and up– and down–arrow keys do not affect the selected text. The left (right) arrow key moves the selection to one character before (after) the current selection.

Normally, written output to a window blocks when the text reaches the end of the screen; a button 2 menu item toggles scrolling.

Other editing operations are selected from a menu on button 2. The cut operation deletes the selected text from the screen and puts it in the snarf buffer; snarf copies the selected text to the buffer without deleting it; paste replaces the selected text with the contents of the buffer; and send copies the snarf buffer to just after the output point, adding a final newline if missing. Paste will sometimes and send will always place text after the output point; the text so placed will behave exactly as described above. Therefore when pasting text containing newlines after the output point, it may be prudent to turn on hold mode first.

The plumber menu item sends the contents of the selection (not the snarf buffer) to the plumber(4). If the selection is empty, it sends the white–space–delimited text containing the selection (typing cursor). A typical use of this feature is to tell the editor to find the source of an error by plumbing the file and line information in a compiler’s diagnostic.

Raw text windows

Opening or manipulating certain files served by r.io suppresses some of the services supplied to ordinary text windows. While the file /dev/mouse is open, any mouse operations are the responsibility of another program running in the window. Thus, r.io refrains from maintaining the scroll bar, supplying text editing or menus, interpreting the VIEW key as a request to scroll, and also turns scrolling on.

The file /dev/consctl controls interpretation of keyboard input. In particular, a raw mode may be set: in a raw–input window, no typed keyboard characters are special, they are not echoed to the screen, and all are passed to a program immediately upon reading, instead of being gathered
into lines.

**Graphics windows**
A program that holds /dev/mouse and /dev/consctl open after putting the console in raw mode has complete control of the window: it interprets all mouse events, gets all keyboard characters, and determines what appears on the screen.

**FILES**
- `/lib/font/bit/*` font directories
- `/mnt/wsys` Files served by rio (also unioned in /dev in a window's name space, before the terminal's real /dev files)
- `/srv/rio.user.pid` Server end of rio.
- `/srv/riowctl.user.pid` Named pipe for wctl messages.

**SOURCE**
- `/sys/src/cmd/rio`
- `/rc/bin/label`
- `/rc/bin/window`
- `/rc/bin/wloc`

**SEE ALSO**
- `rio(4), rc(1), cpu(1), sam(1), mail(1), proof(1), graphics(2), frame(2), window(2), notify(2), cons(3), draw(3), mouse(3), keyboard(6)`

**BUGS**
The standard input of window is redirected to the newly created window, so there is no way to pipe the output of a program to the standard input of the new window. In some cases, `plumb(1)` can be used to work around this limitation.
NAME
   rm – remove files
SYNOPSIS
   rm [−fr] file ... 
DESCRIPTION
   Rm removes files or directories. A directory is removed only if it is empty. Removal of a file
   requires write permission in its directory, but neither read nor write permission on the file itself.
   The options are
   −f    Don’t report files that can’t be removed.
   −r    Recursively delete the entire contents of a directory and the directory itself.
SOURCE
   /sys/src/cmd/rm.c
SEE ALSO
   remove(2)
NAME
rwd, conswdir - maintain remote working directory

SYNOPSIS
rwd path
conswdir [ prog ]

DESCRIPTION
Rwd and conswdir conspire to keep rio(4) and acme(4) informed about the current directory on remote systems during login sessions. Rio and acme include this information in plumb messages sent to plumber(4). If the remote system's name space is mounted in the plumber's name space, the end result is that file paths printed during the session are plumbable.

Rwd informs rio and acme of directory changes. The name of the remote machine is taken from the environment variable $remotesys. Rwd writes the full path to /dev/wdir; writes the last element of the path, suffixed by @remotesys, to /dev/label; and when run inside a win (see acme(1)) window, changes the window title to path/--remotesys using /dev/acme/ctl.

Conswdir copies standard input to standard output, looking for in-band messages about directory changes. The messages are of the form:

\033;path\007
where \033 and \007 are ASCII escape and bell characters. Such messages are removed from the stream and not printed to standard output; for each such message conswdir runs prog (default /bin/rwd) with path as its only argument.

EXAMPLES
Add this plumbing rule (see plumb(6)) in order to run commands in the plumber's name space:

# have plumber run command
kind is text
data matches 'Local (.*)'
plumb to none
plumb start rc -c $1

Mount a Unix system in your name space and the plumber's:

% 9fs unix
% plumb 'Local 9fs unix'

(If you're using acme, execute Local 9fs unix with the middle button to mount the Unix system in acme's name space.)

Connect to the Unix system, processing in-band directory change messages:

% ssh unix | aux/conswdir

Add this shell function to your .profile on the Unix system to generate directory change messages every time a cd command is executed:

H='hostname | sed 's/\..*/\''
_cd () {
\cd $* &&
case $* in
 **)
      _dir='pwd'
      echo /n/$H$_dir | awk '{printf("\033;%s\007", $1);}'
esac
    alias cd=_cd

The examples described so far only help for relative path names. Add this plumbing rule to handle rooted names like /usr/include/stdio.h:

# remote rooted path names
type is text
wdir matches '/n/unix(/.*)?'
data matches '/([.a-zA-Z0-9_\/-]*[a-zA-Z0-9_\/-])('addr')?'
arg isfile /n/unix/$1
data set $file
attr add addr=$3
plumb to edit
plumb client window $editor

SOURCE
/rc/bin/rwd
/sys/src/cmd/aux/conswdir.c

SEE ALSO
plumber(4), plumb(6), srv(4)

BUGS
This mechanism is clunky, but Unix and SSH make it hard to build a better one.
The escape sequence was chosen because it changes the title on xterm windows.
NAME
sam, B, sam.save, samterm - screen editor with structural regular expressions

SYNOPSIS
sam [ option ... ] [ files ]
sam -r machine
sam .save
B [ -nnn ] file ...

DESCRIPTION
Sam is a multi-file editor. It modifies a local copy of an external file. The copy is here called a file. The files are listed in a menu available through mouse button 3 or the n command. Each file has an associated name, usually the name of the external file from which it was read, and a ‘modified’ bit that indicates whether the editor’s file agrees with the external file. The external file is not read into the editor’s file until it first becomes the current file—that to which editing commands apply—whereupon its menu entry is printed. The options are

- a Autoindent. In this mode, when a newline character is typed in the terminal interface, samterm copies leading white space on the current line to the new line.
- d Do not ‘download’ the terminal part of sam. Editing will be done with the command language only, as in ed(1).
- r machine Run the host part remotely on the specified machine, the terminal part locally.
- s path Start the host part from the specified file on the remote host. Only meaningful with the -r option.
- t path Start the terminal part from the specified file. Useful for debugging.

Regular expressions
Regular expressions are as in regexp(6) with the addition of \n to represent newlines. A regular expression may never contain a literal newline character. The empty regular expression stands for the last complete expression encountered. A regular expression in sam matches the longest leftmost substring formally matched by the expression. Searching in the reverse direction is equivalent to searching backwards with the catenation operations reversed in the expression.

Addresses
An address identifies a substring in a file. In the following, ‘character n’ means the null string after the n-th character in the file, with 1 the first character in the file. ‘Line n’ means the n-th match, starting at the beginning of the file, of the regular expression . *\n?. All files always have a current substring, called dot, that is the default address.

Simple Addresses
# n The empty string after character n; #0 is the beginning of the file.
 n Line n; 0 is the beginning of the file.
/ regexp/
? regexp?
The substring that matches the regular expression, found by looking toward the end (/) or beginning (?) of the file, and if necessary continuing the search from the other end to the starting point of the search. The matched substring may straddle the starting point. When entering a pattern containing a literal question mark for a backward search, the question mark should be specified as a member of a class.
0 The string before the first full line. This is not necessarily the null string; see + and – below.
$ The null string at the end of the file.
 . Dot.
' The mark in the file (see the k command below).
"regexp"
 Preceding a simple address (default .), refers to the address evaluated in the unique file whose menu line matches the regular expression.
Compound Addresses
In the following, \texttt{a1} and \texttt{a2} are addresses.

\texttt{a1+a2} The address \texttt{a2} evaluated starting at the end of \texttt{a1}.

\texttt{a1–a2} The address \texttt{a2} evaluated looking in the reverse direction starting at the beginning of \texttt{a1}.

\texttt{a1 , a2} The substring from the beginning of \texttt{a1} to the end of \texttt{a2}. If \texttt{a1} is missing, 0 is substituted. If \texttt{a2} is missing, \$ is substituted.

\texttt{a1 ; a2} Like \texttt{a1 , a2}, but with \texttt{a2} evaluated at the end of, and dot set to, \texttt{a1}.

The operators + and – are high precedence, while , and ; are low precedence.

In both + and – forms, if \texttt{a2} is a line or character address with a missing number, the number defaults to 1. If \texttt{a1} is missing, . is substituted. If both \texttt{a1} and \texttt{a2} are present and distinguishable, + may be elided. \texttt{a2} may be a regular expression; if it is delimited by ?’s, the effect of the + or – is reversed.

It is an error for a compound address to represent a malformed substring. Some useful idioms: \texttt{a1+-} (\texttt{a1--}) selects the line containing the end (beginning) of \texttt{a1}. \texttt{0/regexp/} locates the first match of the expression in the file. (The form \texttt{0; //} sets dot unnecessarily.) \texttt{.//regexp//} finds the second following occurrence of the expression, and \texttt{.//regexp//} extends dot.

Commands
In the following, text demarcated by slashes represents text delimited by any printable character except alphanumerics. Any number of trailing delimiters may be elided, with multiple elisions then representing null strings, but the first delimiter must always be present. In any delimited text, newline may not appear literally; \texttt{\backslash n} may be typed for newline; and \texttt{\backslash} quotes the delimiter, here /. Backslash is otherwise interpreted literally, except in \texttt{s} commands.

Most commands may be prefixed by an address to indicate their range of operation. Those that may not are marked with a * below. If a command takes an address and none is supplied, dot is used. The sole exception is the \texttt{w} command, which defaults to 0, \$. In the description, ‘range’ is used to represent whatever address is supplied. Many commands set the value of dot as a side effect. If so, it is always set to the ‘result’ of the change: the empty string for a deletion, the new text for an insertion, etc. (but see the \texttt{s} and \texttt{e} commands).

Text commands
\texttt{a/text/}
or\texttt{a/}
lines of text
 . Insert the text into the file after the range. Set dot.

\texttt{c}
\texttt{i} Same as \texttt{a}, but \texttt{c} replaces the text, while \texttt{i} inserts before the range.

\texttt{d} Delete the text in the range. Set dot.

\texttt{s/regexp/text/}
Substitute \texttt{text} for the first match to the regular expression in the range. Set dot to the modified range. In text the character & stands for the string that matched the expression. Backslash behaves as usual unless followed by a digit: \texttt{\backslash d} stands for the string that matched the subexpression begun by the \texttt{d}-th left parenthesis. If \texttt{s} is followed immediately by a number \texttt{n}, as in \texttt{s2/x/y/}, the \texttt{n}-th match in the range is substituted. If the command is followed by \texttt{g}, as in \texttt{s/x/y/g}, all matches in the range are substituted.

\texttt{m a1}
\texttt{t a1} Move (m) or copy (t) the range to after \texttt{a1}. Set dot.

Display commands
\texttt{p} Print the text in the range. Set dot.
\texttt{=} Print the line address and character address of the range.
\texttt{##} Print just the character address of the range.

File commands
\texttt{* b file–list}
Set the current file to the first file named in the list that \texttt{sam} also has in its menu. The list
may be expressed <Plan 9 command> in which case the file names are taken as words (in
the shell sense) generated by the Plan 9 command.

* B file-list
  Same as b, except that file names not in the menu are entered there, and all file names in
  the list are examined.

* n
  Print a menu of files. The format is:
  ' or blank indicating the file is modified or clean,
  - or + indicating the file is unread or has been read (in the terminal, * means more
  than one window is open),
  . or blank indicating the current file,
  a blank, 
  and the file name.

* D file-list
  Delete the named files from the menu. If no files are named, the current file is deleted. It
  is an error to D a modified file, but a subsequent D will delete such a file.

I/O Commands

* e filename
  Replace the file by the contents of the named external file. Set dot to the beginning of the
  file.

r filename
  Replace the text in the range by the contents of the named external file. Set dot.

w filename
  Write the range (default 0, $) to the named external file.

* £ filename
  Set the file name and print the resulting menu entry.

If the file name is absent from any of these, the current file name is used. e always sets the file
name; r and w do so if the file has no name.

< Plan 9--command
  Replace the range by the standard output of the Plan 9 command.

> Plan 9--command
  Send the range to the standard input of the Plan 9 command.

| Plan 9--command
  Send the range to the standard input, and replace it by the standard output, of the Plan 9
  command.

* ! Plan 9--command
  Run the Plan 9 command.

* cd directory
  Change working directory. If no directory is specified, $home is used.

In any of <, >, | or !, if the Plan 9 command is omitted the last Plan 9 command (of any type) is
substituted. If sam is downloaded (using the mouse and raster display, i.e. not using option –d),
! sets standard input to /dev/null, and otherwise unassigned output (stdout for ! and >,
stderr for all) is placed in /tmp/sam.err and the first few lines are printed.

Loops and Conditionals

x/regexp/ command
  For each match of the regular expression in the range, run the command with dot set to the
  match. Set dot to the last match. If the regular expression and its slashes are omitted,
  /. */ is assumed. Null string matches potentially occur before every character of the
  range and at the end of the range.

y/regexp/ command
  Like x, but run the command for each substring that lies before, between, or after the
  matches that would be generated by x. There is no default regular expression. Null sub-
  strings potentially occur before every character in the range.

* X/regexp/ command
  For each file whose menu entry matches the regular expression, make that the current file
  and run the command. If the expression is omitted, the command is run in every file.

* Y/regexp/ command
  Same as X, but for files that do not match the regular expression, and the expression is
  required.
Miscellany

k Set the current file’s mark to the range. Does not set dot.
q Quit. It is an error to quit with modified files, but a second q will succeed.
* u n Undo the last n (default 1) top-level commands that changed the contents or name of the current file, and any other file whose most recent change was simultaneous with the current file’s change. Successive u’s move further back in time. The only commands for which u is ineffective are cd, u, q, w and D. If n is negative, u ‘redoes,’ undoing the undo, going forwards in time again.

Grouping and multiple changes

Commands may be grouped by enclosing them in braces { }. Commands within the braces must appear on separate lines (no backslashes are required between commands). Semantically, an opening brace is like a command: it takes an (optional) address and sets dot for each subcommand. Commands within the braces are executed sequentially, but changes made by one command are not visible to other commands (see the next paragraph). Braces may be nested arbitrarily.

When a command makes a number of changes to a file, as in x/re/c/text/, the addresses of all changes to the file are computed in the original file. If the changes are in sequence, they are applied to the file. Successive insertions at the same address are catenated into a single insertion composed of the several insertions in the order applied.

The terminal

What follows refers to behavior of sam when downloaded, that is, when operating as a display editor on a raster display. This is the default behavior; invoking sam with the –d (no download) option provides access to the command language only.

Each file may have zero or more windows open. Each window is equivalent and is updated simultaneously with changes in other windows on the same file. Each window has an independent value of dot, indicated by a highlighted substring on the display. Dot may be in a region not within the window. There is usually a ‘current window’, marked with a dark border, to which typed text and editing commands apply. Text may be typed and edited as in rio(1); also the escape key (ESC) selects (sets dot to) text typed since the last mouse button hit.

The button 3 menu controls window operations. The top of the menu provides the following operators, each of which uses one or more rio-like cursors to prompt for selection of a window or sweeping of a rectangle. ‘Sweeping’ a null rectangle gets a large window, disjoint from the command window or the whole screen, depending on where the null rectangle is.

new Create a new, empty file.
zerox Create a copy of an existing window.
resize As in rio.
close Delete the window. In the last window of a file, close is equivalent to a D for the file.
write Equivalent to a w for the file.

Below these operators is a list of available files, starting with ~sam~, the command window. Selecting a file from the list makes the most recently used window on that file current, unless it is already current, in which case selections cycle through the open windows. If no windows are open on the file, the user is prompted to open one. Files other than ~sam~ are marked with one of the characters –+* according as zero, one, or more windows are open on the file. A further mark . appears on the file in the current window and a single quote, ’, on a file modified since last write.
The command window, created automatically when sam starts, is an ordinary window except that text typed to it is interpreted as commands for the editor rather than passive text, and text printed by editor commands appears in it. The behavior is like rio, with an ‘output point’ that separates commands being typed from previous output. Commands typed in the command window apply to the current open file—the file in the most recently current window.

**Manipulating text**

Button 1 changes selection, much like rio. Pointing to a non–current window with button 1 makes it current; within the current window, button 1 selects text, thus setting dot. Double–clicking selects text to the boundaries of words, lines, quoted strings or bracketed strings, depending on the text at the click.

Button 2 provides a menu of editing commands:

- **cut**  
  Delete dot and save the deleted text in the snarf buffer.

- **paste**  
  Replace the text in dot by the contents of the snarf buffer.

- **snarf**  
  Save the text in dot in the snarf buffer.

- **plumb**  
  Send the text in the selection as a plumb message. If the selection is empty, the white–space–delimited block of text is sent as a plumb message with a click attribute defining where the selection lies (see plumb(6)).

- **look**  
  Search forward for the next occurrence of the literal text in dot. If dot is the null string, the text in the snarf buffer is used. The snarf buffer is unaffected.

- **<rio>**  
  Exchange snarf buffers with rio.

- **/regexp**  
  Search forward for the next match of the last regular expression typed in a command. (Not in command window.)

- **send**  
  Send the text in dot, or the snarf buffer if dot is the null string, as if it were typed to the command window. Saves the sent text in the snarf buffer. (Command window only.)

**External communication**

Sam listens to the edit plumb port. If plumbing is not active, on invocation sam creates a named pipe /srv/sam. user which acts as an additional source of commands. Characters written to the named pipe are treated as if they had been typed in the command window.

B is a shell–level command that causes an instance of sam running on the same terminal to load the named files. B uses either plumbing or the named pipe, whichever service is available. If plumbing is not enabled, the option allows a line number to be specified for the initial position to display in the last named file (plumbing provides a more general mechanism for this ability).

**Abnormal termination**

If sam terminates other than by a q command (by hangup, deleting its window, etc.), modified files are saved in an executable file, $home/sam.save. This program, when executed, asks whether to write each file back to an external file. The answer y causes writing; anything else skips the file.

**FILES**

- $home/sam.save
- $home/sam.err
- /sys/lib/samsave

The program called to unpack $home/sam.save.

**SOURCE**

- /sys/src/cmd/sam  
  source for sam itself
- /sys/src/cmd/samterm  
  source for the separate terminal part
- /rc/bin/B

**SEE ALSO**

- ed(1), sed(1), grep(1), rio(1), regexp(6).

Rob Pike, “The text editor sam”.

202
NAME
aescbc, ipso, seystore - seystore commands

SYNOPSIS
auth/seystore [-cinv] [-(g|G) getfile] [-(p putfile] [-(s server] [-(u user]
auth/aescbc -e [-i] <cleartxt >ciphertext
auth/aescbc -d [-i] <ciphertext >cleartxt
ipso [-a -e -l -f -s] [file ...]

DESCRIPTION
Secstore authenticates to a secure-store server using a password and optionally a hardware token, then saves or retrieves a file. This is intended to be a credentials store (public/private keypairs, passwords, and other secrets) for a factotum.

Option -c prompts for a password change.
Option -g retrieves a file to the local directory; option -G writes it to standard output instead. Specifying getfile of . will send to standard output a list of remote files with dates, lengths and SHA1 hashes.
Option -i says that the password should be read from standard input instead of from /dev/cons.
Option -n says that the password should be read from NVRAM (see authsrv(2)) instead of from /dev/cons.
Option -p stores a file on the seystore.
Option -r removes a file from the seystore.
The server is tcp! $auth! seystore, or the server specified by option -s.
Option -u access the secure-store files belonging to user.
Option -v produces more verbose output, in particular providing a few bits of feedback to help the user detect mistyping.

For example, to add a secret to the file read by factotum(4) at startup, open a new window, type

% ramfs -p; cd /tmp
% auth/seystore -g factotum
seystore password:
% echo 'key proto=apop dom=x.com user=ehg !password=hi' >> factotum
% auth/seystore -p factotum
seystore password:
% read -m factotum > /mnt/factotum/ctl

and delete the window. The first line creates an ephemeral memory-resident workspace, invisible to others and automatically removed when the window is deleted. The next three commands fetch the persistent copy of the secrets, append a new secret, and save the updated file back to seystore. The final command loads the new secret into the running factotum.

The ipso command packages this sequence into a convenient script to simplify editing of files stored on a secure store. It copies the named files into a local ramfs(4) and invokes acme(1) on them. When the editor exits, ipso prompts the user to confirm copying modified or newly created files back to seystore. If no file is mentioned, ipso grabs all the user's files from seystore for editing.

By default, ipso will edit the seystore files and, if one of them is named Factotum, flush current keys from factotum and load the new ones from the file. If the -e, -f, or -l options are given, ipso will just perform only the requested operations, i.e., edit, flush, and/or load.

The -s option of ipso invokes sam(1) as the editor instead of acme; the -a option provides a similar service for files encrypted by aescbc (q.v.). With the -a option, the full rooted pathname of the file must be specified and all files must be encrypted with the same key. Also with -a, newly created files are ignored.
Aescbc encrypts (under \texttt{-e}) and decrypts (under \texttt{-d}) using AES (Rijndael) in cipher block chaining (CBC) mode. Options \texttt{1} and \texttt{n} are as per \texttt{secstore}, except that \texttt{1} reads from file descriptor 3.

\textbf{SOURCE}
\begin{itemize}
\item /rc/bin/ipso
\item /sys/src/cmd/auth/secstore
\end{itemize}

\textbf{SEE ALSO}
\begin{itemize}
\item factotum(4), secstore(8)
\end{itemize}

\textbf{BUGS}

There is deliberately no backup of files on the secstore, so \texttt{-r} (or a disk crash) is irrevocable. You are advised to store important secrets in a second location.

When using \texttt{ipso}, secrets will appear as plain text in the editor window, so use the command in private.
NAME
sed – stream editor

SYNOPSIS
sed [ −n ] [ −g ] [ −e script ] [ −f sfile ] [ file ... ]

DESCRIPTION
Sed copies the named files (standard input default) to the standard output, edited according to a script of commands. The −f option causes the script to be taken from file sfile; these options accumulate. If there is just one −e option and no −f’s, the option −e may be omitted. The −n option suppresses the default output; −g causes all substitutions to be global, as if suffixed g.

A script consists of editing commands, one per line, of the following form:

[ address [, address ] function [ argument ... ] [ ; ]

In normal operation sed cyclically copies a line of input into a pattern space (unless there is something left after a D command), applies in sequence all commands whose addresses select that pattern space, and at the end of the script copies the pattern space to the standard output (except under −n) and deletes the pattern space.

An address is either a decimal number that counts input lines cumulatively across files, a $ that addresses the last line of input, or a context address, /regular-expression/, in the style of regexp(6), with the added convention that \n matches a newline embedded in the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied to non-selected pattern spaces by use of the negation function ! (below).

An argument denoted text consists of one or more lines, all but the last of which end with \ to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line.

An argument denoted rfile or wfile must terminate the command line and must be preceded by exactly one blank. Each wfile is created before processing begins. There can be at most 120 distinct wfile arguments.

a \text Append. Place text on the output before reading the next input line.
b label Branch to the : command bearing the label. If label is empty, branch to the end of the script.
c \text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place text on the output. Start the next cycle.
d Delete the pattern space. Start the next cycle.
D Delete the initial segment of the pattern space through the first newline. Start the next cycle.
g Replace the contents of the pattern space by the contents of the hold space.
G Append the contents of the hold space to the pattern space.
h Replace the contents of the hold space by the contents of the pattern space.
H Append the contents of the pattern space to the hold space.
i \text
text Insert. Place text on the standard output.

n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

N Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

p Print. Copy the pattern space to the standard output.

P Copy the initial segment of the pattern space through the first newline to the standard output.

q Quit. Branch to the end of the script. Do not start a new cycle.

r rfile Read the contents of rfile. Place them on the output before reading the next input line.

s/regular-expression/replacement/flags Substitute the replacement string for instances of the regular-expression in the pattern space. Any character may be used instead of /. For a fuller description see regexp(6). Flags is zero or more of

g Global. Substitute for all non–overlapping instances of the regular expression rather than just the first one.

p Print the pattern space if a replacement was made.

w wfile Write. Append the pattern space to wfile if a replacement was made.

:t label Test. Branch to the : command bearing the label if any substitutions have been made since the most recent reading of an input line or execution of a t. If label is empty, branch to the end of the script.

w wfile Write. Append the pattern space to wfile.

x Exchange the contents of the pattern and hold spaces.

y/string1/string2/ Transform. Replace all occurrences of characters in string1 with the corresponding character in string2. The lengths of string1 and string2 must be equal.

! function Don’t. Apply the function (or group, if function is {) only to lines not selected by the address(es).

# Comment. Ignore the rest of the line.

: label This command does nothing; it bears a label for b and t commands to branch to.

= Place the current line number on the standard output as a line.

{ Execute the following commands through a matching } only when the pattern space is selected.

An empty command is ignored.

EXAMPLES

sed 10q file
Print the first 10 lines of the file.

sed ’/^$/d’
Delete empty lines from standard input.

sed ’s/UNIX/& system/g’
Replace every instance of UNIX by UNIX system.

sed ’s/ */$// drop trailing blanks
/^$/d drop empty lines
s/ */\ replace blanks by newlines
/g
/^$/d’ chapter*
Print the files chapter1, chapter2, etc. one word to a line.

```
nroff -ms manuscript | sed '{
   /^$/p       if last line of file is empty, print it
   //N        if current line is empty, append next line
   /^\n$/D'     if two lines are empty, delete the first
   Delete all but one of each group of empty lines from a formatted manuscript.
}
```

**SOURCE**
/sys/src/cmd/sed.c

**SEE ALSO**
ed(1), grep(1), awk(1), lex(1), sam(1), regexp(6)

**BUGS**
If input is from a pipe, buffering may consume characters beyond a line on which a q command is executed.
NAME
seq – print sequences of numbers

SYNOPSIS
seq [−w] [−f format] [ first [ incr ] ] last

DESCRIPTION
Seq prints a sequence of numbers, one per line, from first (default 1) to as near last as possible, in increments of incr (default 1). The loop is:

for(val = min; val <= max; val += incr) print val;

The numbers are interpreted as floating point.
Normally integer values are printed as decimal integers. The options are
−f format Use the print(2)-style format print for printing each (floating point) number. The default is %g.
−w Equalize the widths of all numbers by padding with leading zeros as necessary. Not effective with option −f, nor with numbers in exponential notation.

EXAMPLES
seq 0 .05 .1
Print 0 0.05 0.1 (on separate lines).
seq −w 0 .05 .1
Print 0.00 0.05 0.10.

SOURCE
/sys/src/cmd/seq.c

BUGS
Option −w always surveys every value in advance. Thus seq −w 1000000000 is a painful way to get an ‘infinite’ sequence.
NAME
   size - print size of executable files

SYNOPSIS
   size [ file ... ]

DESCRIPTION
   Size prints the size of the segments for each of the argument executable files (default v.out).
   The format is

   \[ \text{textsize} + \text{datasize} + \text{bsssize} = \text{total} \]

   where the numbers are in bytes.

SOURCE
   /sys/src/cmd/size.c

SEE ALSO
   a.out(6)
NAME
sleep - suspend execution for an interval

SYNOPSIS
sleep time

DESCRIPTION
Sleep suspends execution for time seconds. Time may be floating-point.

EXAMPLES
Execute a command 100 seconds hence.
{sleep 100; command}&
Repeat a command every 30 seconds.
while (){
    command
    sleep 30
}

SOURCE
/sys/src/cmd/sleep.c

SEE ALSO
sleep(2)
NAME
soelim - preprocess so inclusion commands in troff input

SYNOPSIS
soelim [ files ... ]

DESCRIPTION
Soelim reads the specified files or the standard input and performs the textual inclusion implied by troff(1) directives of the form

    .so some_file

when they appear at the beginning of input lines. This is useful when using programs such as tbl(1) that do not normally do this, allowing placement of individual tables or other text objects in separate files to be run as a part of a large document.

Note that inclusion can be suppressed by using ' instead of . at the start of the line as in:

    'so /usr/share/lib/tmac/tmac.s

SOURCE
/rc/bin/soelim

SEE ALSO
troff(1)

BUGS
The shell script was written by Sape Mullender.
NAME
sort - sort and/or merge files

SYNOPSIS
sort [-cmUmbdfinrwx][ +pos1[−pos2] ... ][ −k pos1[,pos2] ...]
[-o output][−T dir ...][option ...][file ...]

DESCRIPTION
Sort sorts lines of all the files together and writes the result on the standard output. If no input files are named, the standard input is sorted.
The default sort key is an entire line. Default ordering is lexicographic by runes. The ordering is affected globally by the following options, one or more of which may appear:

−M Compare as months. The first three non–white space characters of the field are folded to upper case and compared so that JAN precedes FEB, etc. Invalid fields compare low to JAN.
−b Ignore leading white space (spaces and tabs) in field comparisons.
−d 'Phone directory' order: only letters, accented letters, digits and white space are significant in comparisons.
−f Fold lower case letters onto upper case. Accented characters are folded to their non–accented upper case form.
−i Ignore characters outside the ASCII range 040–0176 in non–numeric comparisons.
−w Like −i, but ignore only tabs and spaces.
−n An initial numeric string, consisting of optional white space, optional plus or minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value.
−g Numbers, like −n but with optional e–style exponents, are sorted by value.
−r Reverse the sense of comparisons.
−tx 'Tab character' separating fields is x.

The notation +pos1 −pos2 restricts a sort key to a field beginning at pos1 and ending just before pos2. Pos1 and pos2 each have the form m.n, optionally followed by one or more of the flags Mbdflgnrz, where m tells a number of fields to skip from the beginning of the line and n tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. A missing .n means .0; a missing −pos2 means the end of the line. Under the −tx option, fields are strings separated by x; otherwise fields are non–empty strings separated by white space. White space before a field is part of the field, except under option −b. A b flag may be attached independently to pos1 and pos2.

The notation −k pos1[,pos2] is how POSIX sort defines fields: pos1 and pos2 have the same format but different meanings. The value of m is origin 1 instead of origin 0 and a missing .n in pos2 is the end of the field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These option arguments are also understood:
−c Check that the single input file is sorted according to the ordering rules; give no output unless the file is out of sort.
−m Merge; assume the input files are already sorted.
−u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.
−o The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.
−Tdir  Put temporary files in dir rather than in /tmp.

EXAMPLES

sort −u +0f +0 list
Print in alphabetical order all the unique spellings in a list of words where capitalized words differ from uncapitalized.

sort −t: +1 /adm/users
Print the users file sorted by user name (the second colon-separated field).

sort −umM dates
Print the first instance of each month in an already sorted file. Options −um with just one input file make the choice of a unique representative from a set of equal lines predictable.

grep −n ‘^’ input | sort −t: +1f +0n | sed ‘s/\[0−9\]*://’
A stable sort: input lines that compare equal will come out in their original order.

FILES
/tmp/sort.<pid>.<ordinal>

SOURCE
/sys/src/cmd/sort.c

SEE ALSO
uniq(1), look(1)

DIAGNOSTICS
Sort comments and exits with non-null status for various trouble conditions and for disorder discovered under option −C.

BUGS
An external null character can be confused with an internally generated end-of-field character. The result can make a sub-field not sort less than a longer field.
Some of the options, e.g. −i and −M, are hopelessly provincial.
NAME
  spell, sprog - find spelling errors

SYNOPSIS
  spell [ options ] ... [ file ] ...
  aux/sprog [ options ] [−f file ]

DESCRIPTION
  Spell looks up words from the named files (standard input default) in a spelling list and places possible misspellings—words not sanctioned there—on the standard output.
  Spell ignores constructs of troff(1) and its standard preprocessors. It understands these options:
  −b Check British spelling.
  −v Print all words not literally in the spelling list, with derivations.
  −x Print on standard error, marked with =, every stem as it is looked up in the spelling list, along with its affix classes.

As a matter of policy, spell does not admit multiple spellings of the same word. Variants that follow general rules are preferred over those that don’t, even when the unruly spelling is more common. Thus, in American usage, ‘modelled’, ‘sizeable’, and ‘judgment’ are rejected in favor of ‘modeled’, ‘sizable’, and ‘judgement’. Agglutinated variants are shunned; ‘crewmember’ and ‘backyard’ cede to ‘crew member’ and ‘back yard’ (noun) or ‘back–yard’ (adjective).

FILES
  /sys/lib/amspell American spelling list
  /sys/lib/brspell British spelling list
  /bin/aux/sprog The actual spelling checker. It expects one word per line on standard input, and takes the same arguments as spell.

SOURCE
  /rc/bin/spell the script
  /sys/src/cmd/spell source for sprog

SEE ALSO
  deroff(1)

BUGS
  The heuristics of deroff(1) used to excise formatting information are imperfect.
  The spelling list’s coverage is uneven; in particular biology, medicine, and chemistry, and perforce proper names, not to mention languages other than English, are covered very lightly.
NAME
spin – verification tool for models of concurrent systems

SYNOPSIS
spin –a [-m][-Pcpp] file
spin [-bglmprsv][-nN][-Pcpp] file
spin –c [-t][-Pcpp] file
spin –d [-Pcpp] file
spin –f ltl
spin –F file
spin –i [-bglmprsv][-nN][-Pcpp] file
spin –M [-t][-Pcpp] file
spin –t[N][-bglmprsv][-jN][-Pcpp] file
spin –V

DESCRIPTION
Spin is a tool for analyzing the logical consistency of asynchronous systems, specifically distributed software and communication protocols. A verification model of the system is first specified in a guarded command language called Promela. This specification language, described in the reference, allows for the modeling of dynamic creation of asynchronous processes, non-deterministic case selection, loops, goto's, local and global variables. It also allows for a concise specification of logical correctness requirements, including, but not restricted to requirements expressed in linear temporal logic.

Given a Promela model stored in file, spin can perform interactive, guided, or random simulations of the system's execution. It can also generate a C program that performs an exhaustive or approximate verification of the correctness requirements for the system.

–a Generate a verifier (model checker) for the specification. The output is written into a set of C files, named pan.[cbhmt], that can be compiled (pcc pan.c) to produce an executable verifier. The online spin manuals (see below) contain the details on compilation and use of the verifiers.

–c Produce an ASCII approximation of a message sequence chart for a random or guided (when combined with –t) simulation run. See also option –M.

–d Produce symbol table information for the model specified in file. For each Promela object this information includes the type, name and number of elements (if declared as an array), the initial value (if a data object) or size (if a message channel), the scope (global or local), and whether the object is declared as a variable or as a parameter. For message channels, the data types of the message fields are listed. For structure variables, the third field defines the name of the structure declaration that contains the variable.

–f ltl Translate the LTL formula ltl into a never claim.
This option reads a formula in LTL syntax from the second argument and translates it into Promela syntax (a never claim, which is Promela’s equivalent of a Büchi Automaton). The LTL operators are written: [] (always), <> (eventually), and U (strong until). There is no X (next) operator, to secure compatibility with the partial order reduction rules that are applied during the verification process. If the formula contains spaces, it should be quoted to form a single argument to the spin command.

–F file Translate the LTL formula stored in file into a never claim.
This behaves identically to option –f but will read the formula from the file instead of from the command line. The file should contain the formula as the first line. Any text that follows this first line is ignored, so it can be used to store comments or annotation on the formula. (On some systems the quoting conventions of the shell complicate the use of option –F. Option –F is meant to solve those problems.)
-i Perform an interactive simulation, prompting the user at every execution step that requires a nondeterministic choice to be made. The simulation proceeds without user intervention when execution is deterministic.

-M Produce a message sequence chart in Postscript form for a random simulation or a guided simulation (when combined with -t), for the model in file, and write the result into file.ps. See also option -c.

-m Changes the semantics of send events. Ordinarily, a send action will be (blocked) if the target message buffer is full. With this option a message sent to a full buffer is lost.

-nN Set the seed for a random simulation to the integer value N. There is no space between the -n and the integer N.

-t Perform a guided simulation, following the error trail that was produces by an earlier verification run, see the online manuals for the details on verification.

-V Prints the spin version number and exits.

With only a filename as an argument and no options, spin performs a random simulation of the model specified in the file (standard input is the default if the filename is omitted). If option -i is added, the simulation is interactive, or if option -t is added, the simulation is guided.

The simulation normally does not generate output, except what is generated explicitly by the user within the model with printf statements, and some details about the final state that is reached after the simulation completes. The group of options -bg1mprsv sets the desired level of information that the user wants about a random, guided, or interactive simulation run. Every line of output normally contains a reference to the source line in the specification that generated it.

-b Suppress the execution of printf statements within the model.

-g Show at each time step the current value of global variables.

-l In combination with option -p, show the current value of local variables of the process.

-p Show at each simulation step which process changed state, and what source statement was executed.

-r Show all message-receive events, giving the name and number of the receiving process and the corresponding the source line number. For each message parameter, show the message type and the message channel number and name.

-s Show all message-send events.

-vVerbose mode, add some more detail, and generate more hints and warnings about the model.

SOURCE
/sys/src/cmd/spin

SEE ALSO
NAME
  split - split a file into pieces

SYNOPSIS
  split [ option ...] [file]

DESCRIPTION
  Split reads file (standard input by default) and writes it in pieces of 1000 lines per output file. The
  names of the output files are xaa, xab, and so on to xzz. The options are
  -n n  Split into n-line pieces.
  -l n  Synonym for -n n, a nod to Unix’s syntax.
  -e expression
    File divisions occur at each line that matches a regular expression; see regexp(6). Multiple
    -e options may appear. If a subexpression of expression is contained in parentheses
    (.), the output file name is the portion of the line which matches the subexpression.
  -f stem
    Use stem instead of x in output file names.
  -s suffix
    Append suffix to names identified under -e.
  -x
    Exclude the matched input line from the output file.
  -i
    Ignore case in option -e; force output file names (excluding the suffix) to lower case.

SOURCE
  /sys/src/cmd/split.c

SEE ALSO
  sed(1), awk(1), grep(1), regexp(6)
NAME
    src - find source code for executable

SYNOPSIS
    src [−n][−s symbol] file ...

DESCRIPTION
    Src examines the named files to find the corresponding source code, which is then sent to the editor using B (see sam(1)). If file is an rc(1) script, the source is the file itself. If file is an executable, the source is defined to be the single file containing the definition of main and src will point the editor at the line that begins the definition. Src uses db(1) to extract the symbol table information that identifies the source.

    Src looks for each file in the current directory, in /bin, and in the subdirectories of /bin, in that order.

    The −n flag causes src to print the file name but not send it to the editor. The −s flag identifies a symbol other than main to locate.

EXAMPLES
    Find the source to the main routine in /bin/ed:
        src ed
    Find the source for strcmp:
        src −s strcmp rc

SOURCE
    /rc/bin/src

SEE ALSO
    db(1), plum(1), sam(1).
NAME
  ssh, sshnet, scp, sshserve - secure login and file copy from/to Unix or Plan 9
SYNOPSIS
  ssh [-CfiLmpPrw] [-A authlist] [-c cipherlist] [-lu] [user@]host [:cmd [:args ...]]
  sshnet [-A authlist] [-c cipherlist] [-m mtpt] [-s service] [user@]host
  scp [host:]file [host:]file
  scp [host:]file ... [host:]dir
  aux/sshserve [-p] address
DESCRIPTION
  Ssh allows authenticated login over an encrypted channel to hosts that support the ssh protocol
  (see the RFCs listed below for encryption and authentication details).
  Ssh takes the host name of the machine to connect to as its mandatory argument. It may be specified
  as a domain name or an IP address. Normally, login is attempted using the user name from
  /dev/user.
  Command-line options are:
  -C   force input to be read in cooked mode: "line at a time" with local echo.
  -f   enable agent forwarding. With this flag, ssh uses SSH's agent forwarding protocol to allow
       programs running on the remote server to interact with factotum(4) to perform RSA authen-
       tication.
  -i   force interactive mode. In interactive mode, ssh prompts for passwords and confirmations
       of new host keys when necessary. (In non-interactive mode, password requests are
       rejected and unrecognized host keys are cause for disconnecting.) By default, ssh runs in
       interactive mode only when its input file descriptor is /dev/cons.
  -I   force non-interactive mode.
  -m   disable the control-
  -p   force pseudoterminal request. The ssh protocol, grounded in Unix tradition, differentiates
       between connections that request controlling pseudoterminals and those that do not. 
       By default, ssh requests a pseudoterminal only when no command is given.
  -P   force no pseudoterminal request.
  -r   strip carriage returns.
  -R   put the allocated pseudoterminal, if any, in raw mode.
  -w   notify the remote side whenever the window changes size.
  -[lu] user
       specify user name. This option is deprecated in favor of the user@hostname syntax.
  -A authlist
       specify an ordered space-separated list of authentication protocols to try. The full set of
       authentication protocols is rsa (RSA using factotum(4) to moderate key usage), 
       password (use a password gathered from factotum), and tis (challenge-response). The
       default list is all three in that order.
  -c cipherlist
       specify an ordered space-separated list of allowed ciphers to use when encrypting the
       channel. The full set of ciphers is des (standard DES), 3des (a somewhat doubtful vari-
       ation on triple DES), blowfish (Bruce Schneier’s Blowfish), rc4 (RC4), and none (no
       encryption). The default cipher list is blowfish rc4 3des.
  The control-
character is a local escape, as in con(1). It prompts with >>>. Legitimate responses
  to the prompt are
  q Exit.
. Return from the escape.
!cmd Run the command with the network connection as its standard input and output. Standard error will go to the screen.
 r Toggle printing of carriage returns.

If no command is specified, a login session is started on the remote host. Otherwise, the command is executed with its arguments.

`Ssh` establishes a connection with an ssh daemon on the remote host. The daemon sends to `ssh` its RSA public host key and session key. Using these, `ssh` sends a session key which, presumably, only the daemon can decipher. After this, both sides start encrypting their data with this session key.

When the daemon’s host key has been received, `ssh` looks it up in `$HOME/lib/keyring` and in `/sys/lib/ssh/keyring`. If the key is found there, and it matches the received key, `ssh` is satisfied. If not, `ssh` reports this and offers to add the key to `$HOME/lib/keyring`.

Over the encrypted channel, `ssh` attempts to convince the daemon to accept the call using the listed authentication protocols (see the –A option above).

The preferred way to authenticate is a `netkey`-style challenge/response or via a SecurID token. `Ssh` users on other systems than Plan 9 should enable `tis` authentication.

When the connection is authenticated, the given command line, (by default, a login shell) is executed on the remote host.

The SSH protocol allows clients to make outgoing TCP calls via the server. `Sshnet` establishes an SSH connection and, rather than execute a remote command, presents the remote server’s TCP stack as a network stack (see the discussion of TCP in `ip(3)` mounted at `mtpt` (default `/net`), optionally posting a 9P service descriptor for the new file system as `/srv/service`. The –A and –c arguments are as in `ssh`.

`Scp` uses `ssh` to copy files from one host to another. A remote file is identified by a host name, a colon and a file name (no spaces). `Scp` can copy files from remote hosts and to remote hosts.

`SshServe` is the server that services `ssh` calls from remote hosts. The –A and –c options set valid authentication methods and ciphers as in `ssh`, except that there is no rsa authentication method. Unlike in `ssh`, the list is not ordered: the server presents a set and the client makes the choice. The default sets are `tis` and blowfish rc4 3des. By default, users start with the namespace defined in `/lib/namespace`. Users in group `noworld` in `/adm/users` start with the namespace defined in `/lib/namespace.noworld`. `SshServe` does not provide the TCP forwarding functionality used by `sshnet`, because many Unix clients present this capability in an insecure manner.

`SshServe` requires that `factotum(4)` hold the host key, identified by having attributes proto=rsa service=sshserve. To generate a host key:

```
auth/rsagen -t 'service=sshserve' >/mnt/factotum/ctl
```

To extract the public part of the host key in the form used by SSH key rings:

```
grep 'service=sshserve' /mnt/factotum/ctl | auth/rsa2ssh
```

**FILES**

/sys/lib/ssh/keyring
System key ring file containing public keys for remote ssh clients and servers.

/usr/user/lib/keyring
Personal key ring file containing public keys for remote ssh clients and servers.

**SOURCE**

/sys/src/cmd/ssh

**SEE ALSO**

/lib/rfc/rfc425[0–6]
`factotum(4), authsrv(6), rsa(8)`
BUGS

Only version 1 of the SSH protocol is implemented.
NAME
    stop, start - print commands to stop and start processes

SYNOPSIS
    stop name
    start name

DESCRIPTION
    Stop prints commands that will cause all processes called name and owned by the current user to be stopped. The processes can then be debugged when they are in a consistent state.
    Start prints commands that will cause all stopped processes called name and owned by the current user to be started again.
    Use the send command of rio(1), or pipe into rc(1) to execute the commands.

SOURCE
    /rc/bin/stop
    /rc/bin/start

SEE ALSO
    ps(1), kill(1), proc(3)
NAME
   strings - extract printable strings

SYNOPSIS
   strings [-m min][ file ...]

DESCRIPTION
   Strings finds and prints strings containing min (default 6) or more consecutive printable UTF-encoded characters in a (typically) binary file, default standard input. Printable characters are taken to be ASCII characters from blank through tilde (hexadecimal 20 through 7E), inclusive, and all other characters from value 00A0 to FFFF. Strings reports the decimal offset within the file at which the string starts and the text of the string. If the string is longer than 70 runes the line is terminated by three dots and the printing is resumed on the next line with the offset of the continuation line.

SOURCE
   /sys/src/cmd/strings.c

SEE ALSO
   nm(1)
NAME
strip - remove symbols from binary files

SYNOPSIS
strip file ...
strip -o ofile file

DESCRIPTION
Strip removes symbol table segments from executable files, rewriting the files in place. Stripping a file requires write permission of the file and the directory it is in.

If the –o flag is given, the single input file file is stripped and the result written to ofile. File is unchanged.

SOURCE
/sys/src/cmd/strip.c

SEE ALSO
a.out(6)
NAME
sum, md5sum, shalsum - sum and count blocks in a file

SYNOPSIS
sum [ −5r ] [ file ... ]
md5sum [ file ... ]
shalsum [ −2 bits ] [ file ... ]

DESCRIPTION
By default, sum calculates and prints a 32–bit hexadecimal checksum, a byte count, and the name of each file. The checksum is also a function of the input length. If no files are given, the standard input is summed. Other summing algorithms are available. The options are

−r Sum with the algorithm of System V’s sum −r and print the length (in 1K blocks) of the input.
−5 Sum with System V’s default algorithm and print the length (in 512–byte blocks) of the input.

Sum is typically used to look for bad spots, to validate a file communicated over some transmission line or as a quick way to determine if two files on different machines might be the same.

Md5sum computes the 32 hex digit RSA Data Security, Inc. MD5 Message–Digest Algorithm described in RFC1321.

Shalsum computes the 40 hex digit National Institute of Standards and Technology (NIST) SHA1 secure hash algorithm described in FIPS PUB 180–1, by default. Given the 2 option, it instead computes the bits–bit NIST SHA2 secure hash algorithm described in FIPS PUB 180–2 and prints the hash in hex. Currently supported values of bits are 224, 256, 384, and 512.

SOURCE
/sys/src/cmd/sum.c
/sys/src/cmd/md5sum.c
/sys/src/cmd/shalsum.c

SEE ALSO
cmp(1), wc(1), sechash(2)
NAME
syscall - test a system call

SYNOPSIS
syscall [−osx ] entry [ arg ... ]

DESCRIPTION
Syscall invokes the system call entry with the given arguments. (Some functions, such as write and read(2), although not strictly system calls, are valid entries.) It prints the return value and the error string, if there was an error. An argument is either an integer constant as in C (its value is passed), a string (its address is passed), or the literal buf (a pointer to a 1MB buffer is passed).

If −o is given, the contents of the 1MB buffer are printed as a zero-terminated string after the system call is done. The −X and −s options are similar, but −X formats the data as hexadecimal bytes, while −s interprets the data as a stat(5) message and formats it similar to the style of ls and lqm (see ls(1)), with extra detail about the modify and access times.

EXAMPLES
Write a string to standard output:
   syscall write 1 hello 5

Print information about the file connected to standard input:
   syscall −s fstat 0 buf 1024

SOURCE
/sys/src/cmd/syscall

SEE ALSO
Section 2 of this manual.

DIAGNOSTICS
If entry is not known to syscall, the exit status is unknown. If the system call succeeds, the exit status is null; otherwise the exit status is the string that errstr(2) returns.
NAME
tagrd - plumb a Mifare Ultralight tag

SYNOPSIS
tagrd [-D] /dev/cci*/rpc

DESCRIPTION
Tagrd runs continuously, polling a usb(4) ccid-based Touchatag reader and plumbs a message for any tag with its content and its UID. It plumbs another message when the tag disappears. For an example of how to use this, see /sys/src/cmd/scard/plumbing.

When the program successfully communicates with the reader, its led should turn orange.

This program is part of an ongoing programming of the ISO smartcard standards, part of the library can be seen at /sys/src/cmd/scard.

SOURCE
/sys/src/cmd/scard/tagrd.c
/sys/src/cmd/scard/plumbing

SEE ALSO
usb(4)
NAME
tail - deliver the last part of a file

SYNOPSIS
tail [ +–number[lbc][rf]] [ file ]
tail [ –fr ] [ –n nlines ] [ –c nbytes ] [ file ]

DESCRIPTION
Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is copied.

Copying begins at position +number measured from the beginning, or –number from the end of the input. Number is counted in lines, 1K blocks or bytes, according to the appended flag l, b, or c. Default is –101 (ten ell).

The further flag r causes tail to print lines from the end of the file in reverse order; f (follow) causes tail, after printing to the end, to keep watch and print further data as it appears.

The second syntax is that promulgated by POSIX, where the numbers rather than the options are signed.

EXAMPLES
tail file
   Print the last 10 lines of a file.
tail +0f file
   Print a file, and continue to watch data accumulate as it grows.
sed 10q file
   Print the first 10 lines of a file.

SOURCE
/sys/src/cmd/tail.c

BUGS
Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length.
According to custom, option +number counts lines from 1, and counts blocks and bytes from 0. Tail is ignorant of UTF.
NAME
tar, dircp - archiver

SYNOPSIS
tar key [ file ... ]
dircp fromdir todir

DESCRIPTION
Tar saves and restores file trees. It is most often used to transport a tree of files from one system to another. The key is a string that contains at most one function letter plus optional modifiers. Other arguments to the command are names of files or directories to be dumped or restored. A directory name implies all the contained files and subdirectories (recursively).

The function is one of the following letters:

c  Create a new archive with the given files as contents.

r  The named files are appended to the archive.

t  List all occurrences of each file in the archive, or of all files if there are no file arguments.

x  Extract the named files from the archive. If a file is a directory, the directory is extracted recursively. Modes are restored if possible. If no file argument is given, extract the entire archive. If the archive contains multiple entries for a file, the latest one wins.

The modifiers are:

f  Use the next argument as the name of the archive instead of the default standard input (for keys x and t) or standard output (for keys c and r).

g  Use the next (numeric) argument as the group id for files in the output archive.

i  Ignore errors encountered when reading. Errors writing either produce a corrupt archive or indicate deeper file system problems.

k  (keep) Modifies the behavior of x not to extract files which already exist.
m  Do not set the modification time on extracted files. This is the default behavior; the flag exists only for compatibility with other tars.

p  Create archive in POSIX ustar format, which raises the maximum pathname length from 100 to 256 bytes. Ustar archives are recognised automatically by tar when reading archives. This is the default behavior; the flag exists only for backwards compatibility with older versions of tar.

P  Do not generate the POSIX ustar format.

R  When extracting, respect leading slash on file names. By default, files are always extracted relative to the current directory.

s  When extracting, attempt to resynchronise after not finding a tape header block where expected.

T  Modifies the behavior of x to set the modified time, mode and, for POSIX archives and filesystem permitting, the user and group of each file to that specified in the archive.

u  Use the next (numeric) argument as the user id for files in the output archive. This is only useful when moving files to a non–Plan 9 system.

v  (verbose) Print the name of each file as it is processed. With t, give more details about the archive entries.

z  Operate on compressed tar archives. The type of compression is inferred from the file name extension: gzip(1) for .tar.gz and .tgz; bzip2 (see gzip(1)) for .tar.bz, .tbz, .tar.bz2, and .tbz2; compress for .tar.Z and .tz. If no extension matches, gzip is used. The z flag is unnecessary (but allowed) when using the t and x verbs on archives with recognized extensions.
EXAMPLES

Tar can be used to copy hierarchies thus:

@{cd fromdir && tar c .} | @{cd todir && tar xT}

Dircp does this.

SOURCE
/sys/src/cmd/tar.c
/rc/bin/dircp

SEE ALSO
ar(1), bundle(1), tapefs(4), mkfs(8)

BUGS

There is no way to ask for any but the last occurrence of a file.

File path names are limited to 100 characters (256 when using ustar format).

The tar format allows specification of links and symbolic links, concepts foreign to Plan 9: they are ignored.

The r key (append) cannot be used on compressed archives.

Tar, thus dircp, doesn’t record Plan-9-specific metadata such as append-only and exclusive-open permission bits, so they aren’t copied.
NAME
   tbl - format tables for nroff or troff

SYNOPSIS
   tbl [ file ]

DESCRIPTION
   *Tb* 1 is a preprocessor for formatting tables for *nroff* or *troff(1)*. The input *files* are copied to the standard output, except for segments of the form
     .TS
     options ;
     format .
     data
     .T&
     format .
     data
     ...
     .TE
which describe tables and are replaced by *troff* requests to lay out the tables. If no arguments are given, *tbl* reads the standard input.

The (optional) *options* line is terminated by a semicolon and contains one or more of

- `center` center the table; default is left-adjust
- `expand` make table as wide as current line length
- `box` enclose the table in a box or double box
- `doublebox` enclose every item in a box
- `tab(x)` use *x* to separate input items; default is tab
- `linesize(n)` set rules in *n*-point type
- `delim(xy)` recognize *x* and *y* as *eqn(1)* delimiters

Each line, except the last, of the obligatory *format* describes one row of the table. The last line describes all rows until the next . T&; where the format changes, or the end of the table at . TE. A format is specified by key letters, one per column, either upper or lower case:

- `L` Left justify: the default for columns without format keys.
- `R` Right justify.
- `C` Center.
- `N` Numeric: align at decimal point (inferred for integers) or at \&.
- `S` Span: extend previous column across this one.
- `A` Alphabetic: left–aligned within column, widest item centered, indented relative to L rows.
- `^` Vertical span: continue item from previous row into this row.
- `=` Draw a horizontal rule in this column.
- `=` Draw a double horizontal rule in this column.

Key letters may be followed by modifiers, also either case:

- `|` Draw vertical rule between columns.
- `||` Draw a double vertical rule between columns.
- `n` Gap between column is *n* ens wide. Default is 3.
- `Ffont` Use specified *font*. B and I mean FB and FI.
- `T` Begin vertically–spanned item at top row of range; default is vertical centering (with ^).
- `Pn` Use point size *n*.
- `Vn` Use *n*-point vertical spacing in text block; signed *n* means relative change.
- `W(n)` Column width as a *troff* width specification. Parentheses are optional if *n* is a simple integer.
- `E` Equalize the widths of all columns marked E.

Each line of *data* becomes one row of the table; tabs separate items. Lines beginning with . are *troff* requests. Certain special data items are recognized:

231
<table>
<thead>
<tr>
<th>Town</th>
<th>Households</th>
<th>Number</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bedminster</td>
<td>789</td>
<td>3.26</td>
<td></td>
</tr>
<tr>
<td>Bernards Twp.</td>
<td>3087</td>
<td>3.74</td>
<td></td>
</tr>
<tr>
<td>Bernardsville</td>
<td>2018</td>
<td>3.30</td>
<td></td>
</tr>
</tbody>
</table>

When it is used in a pipeline with `eqn`, the `tbl` command should be first, to minimize the volume of data passed through pipes.

**EXAMPLES**

Let `<tab>` represent a tab (which should be typed as a genuine tab).

```plaintext
.TS
 c s s
 c c s
 c c c

<table>
<thead>
<tr>
<th>Household Population</th>
<th>Town</th>
<th>Number</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Town</td>
<td>Households</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bedminster</td>
<td>789</td>
<td>3.26</td>
<td></td>
</tr>
<tr>
<td>Bernards Twp.</td>
<td>3087</td>
<td>3.74</td>
<td></td>
</tr>
<tr>
<td>Bernardsville</td>
<td>2018</td>
<td>3.30</td>
<td></td>
</tr>
</tbody>
</table>

.SOURCE
/sys/src/cmd/tbl

SEE ALSO

troff(1), eqn(1), doctype(1)

NAME
tcs – translate character sets

SYNOPSIS
tcs [−s] [−f ics] [−t ocs] [file ...]

DESCRIPTION
Tcs interprets the named file(s) (standard input default) as a stream of characters from the ics character set or format, converts them to runes, and then converts them into a stream of characters from the ocs character set or format on the standard output. The default value for ics and ocs is utf, the UTF encoding described in utf(6). The −l option lists the character sets known to tcs. Processing continues in the face of conversion errors (the −s option prevents reporting of these errors). The −c option forces the output to contain only correctly converted characters; otherwise, Runeerror (0xFFFD) characters will be substituted for UTF encoding errors and unknown characters.

The −v option generates various diagnostic and summary information on standard error, or makes the −l output more verbose.

Tcs recognizes an ever changing list of character sets. In particular, it supports a variety of Russian and Japanese encodings. Some of the supported encodings are:

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>utf</td>
<td>The Plan 9 UTF encoding, known by ISO as UTF-8</td>
</tr>
<tr>
<td>utf1</td>
<td>The deprecated original UTF encoding from ISO 10646</td>
</tr>
<tr>
<td>ascii</td>
<td>7-bit ASCII</td>
</tr>
<tr>
<td>8859–1</td>
<td>Latin–1 (Central European)</td>
</tr>
<tr>
<td>8859–2</td>
<td>Latin–2 (Czech .. Slovak)</td>
</tr>
<tr>
<td>8859–3</td>
<td>Latin–3 (Dutch .. Turkish)</td>
</tr>
<tr>
<td>8859–4</td>
<td>Latin–4 (Scandinavian)</td>
</tr>
<tr>
<td>8859–5</td>
<td>Part 5 (Cyrillic)</td>
</tr>
<tr>
<td>8859–6</td>
<td>Part 6 (Arabic)</td>
</tr>
<tr>
<td>8859–7</td>
<td>Part 7 (Greek)</td>
</tr>
<tr>
<td>8859–8</td>
<td>Part 8 (Hebrew)</td>
</tr>
<tr>
<td>8859–9</td>
<td>Latin–5 (Finnish .. Portuguese)</td>
</tr>
<tr>
<td>html</td>
<td>Unicode as encoded by HTML</td>
</tr>
<tr>
<td>koi8</td>
<td>KOI–8 (GOST 19769–74)</td>
</tr>
<tr>
<td>jis–kanji</td>
<td>ISO 2022–JP</td>
</tr>
<tr>
<td>ujis</td>
<td>EUC–JX: JIS 0208</td>
</tr>
<tr>
<td>ms–kanji</td>
<td>Microsoft, or Shift–JIS</td>
</tr>
<tr>
<td>jis</td>
<td>(from only) guesses between ISO 2022–JP, EUC or Shift–Jis</td>
</tr>
<tr>
<td>gb</td>
<td>Chinese national standard (GB2312–80)</td>
</tr>
<tr>
<td>big5</td>
<td>Big 5 (HKU version)</td>
</tr>
<tr>
<td>unicode</td>
<td>Unicode Standard 1.0</td>
</tr>
<tr>
<td>tis</td>
<td>Thai character set plus ASCII (TIS 620–1986)</td>
</tr>
<tr>
<td>msdos</td>
<td>IBM PC: CP 437</td>
</tr>
<tr>
<td>atari</td>
<td>Atari–ST character set</td>
</tr>
</tbody>
</table>

EXAMPLES
tcs −f 8859–1
Convert 8859–1 (Latin–1) characters into UTF format.

tcs −s −f jis
Convert characters encoded in one of several shift JIS encodings into UTF format. Unknown Kanji will be converted into 0xFFFD characters.

tcs −t html
Convert UTF into character set–independent HTML.

tcs −lv
Print an up to date list of the supported character sets.

SOURCE
/sys/src/cmd/tcs


SEE ALSO

 ascii(1), rune(2), utf(6).
NAME
  tee - pipe fitting

SYNOPSIS
  tee [-i][-a] files

DESCRIPTION
  *Tee* transcribes the standard input to the standard output and makes copies in the *files*. The options are
  
  -i  Ignore interrupts.
  
  -a  Append the output to the *files* rather than rewriting them.

SOURCE
  /sys/src/cmd/tee.c
NAME
tel, iwhois - look in phone book

SYNOPSIS
tel key ...
iwhois name[@domain]

DESCRIPTION
Tel looks up key in a private telephone book, $home/lib/tel, and in the public telephone book, /lib/tel. It uses grep (with the -i option to ignore case differences), so the key may be any part of a name or number. Customarily, the telephone book contains names, userids, home numbers, and office numbers of users. It also contains a directory of area codes and miscellaneous people of general interest.

Iwhois looks up names in the Internet NIC's personnel database. Name should be a surname optionally followed by a comma and given name. A different server can be chosen by appending to the name an @ followed by the server's domain name.

FILES
/lib/areacodes Telephone area codes database.
/lib/tel Public telephone number database.
$home/lib/tel Personal telephone number database.

SOURCE
/rc/bin/tel
/rc/bin/iwhois
NAME
test – set status according to condition

SYNOPSIS
test expr

DESCRIPTION
Test evaluates the expression expr. If the value is true the exit status is null; otherwise the exit status is non-null. If there are no arguments the exit status is non-null.

The following primitives are used to construct expr.

-`r file` True if the file exists (is accessible) and is readable.
-`w file` True if the file exists and is writable.
-`x file` True if the file exists and has execute permission.
-`e file` True if the file exists.
-`f file` True if the file exists and is a plain file.
-`d file` True if the file exists and is a directory.
-`s file` True if the file exists and has a size greater than zero.
-`t fildes` True if the open file whose file descriptor number is fildes (1 by default) is the same file as /dev/cons.
-`A file` True if the file exists and is append-only.
-`L file` True if the file exists and is exclusive-use.
-`T file` True if the file exists and is temporary.
-`s1 = s2` True if the strings s1 and s2 are identical.
-`s1 != s2` True if the strings s1 and s2 are not identical.
-`s1` True if s1 is not the null string. (Deprecated.)
-`n s1` True if the length of string s1 is non-zero.
-`z s1` True if the length of string s1 is zero.

-`nl -eq n2` True if the integers nl and n2 are arithmetically equal. Any of the comparisons `-ne`, `-gt`, `-ge`, `-lt`, or `-le` may be used in place of `-eq`. The (nonstandard) construct `-l string`, meaning the length of string, may be used in place of an integer.

-`a -nt b` True if file a is newer than (modified after) file b.
-`a -ot b` True if file a is older than (modified before) file b.
-`f -older t` True if file f is older than (modified before) time t. If t is a integer followed by the letters y(years), M(months), d(days), h(hours), m(minutes), or s(seconds), it represents current time minus the specified time. If there is no letter, it represents seconds since epoch. You can also concatenate mixed units. For example, 3d12h means three days and twelve hours ago.

These primaries may be combined with the following operators:

`!` unary negation operator
-o binary or operator
-a binary and operator; higher precedence than -o
(  expr  ) parentheses for grouping.

The primitives `-b`, `-u`, `-g`, and `-s` return false; they are recognized for compatibility with POSIX.

Notice that all the operators and flags are separate arguments to test. Notice also that parentheses and equal signs are meaningful to rc and must be enclosed in quotes.

EXAMPLES
Test is a dubious way to check for specific character strings: it uses a process to do what an rc(1) match or switch statement can do. The first example is not only inefficient but wrong, because test understands the purported string "-c" as an option.

    if (test $1 '="-c") echo OK # wrong!

A better way is

    if (~ $1 -c) echo OK

Test whether abc is in the current directory.
test -f abc -o -d abc

SOURCE
/sys/src/cmd/test.c

SEE ALSO
rc(1)

BUGS
Won't complain about extraneous arguments since there may be arguments left unprocessed by short-circuit evaluation of -a or -o.
NAME
  thesaurus - search online thesaurus

SYNOPSIS
  thesaurus word

DESCRIPTION
  *thesaurus* searches the online thesaurus at http://thesaurus.reference.com

SOURCE
  /rc/bin/thesaurus
NAME
time - time a command

SYNOPSIS
time command [ arg ... ]

DESCRIPTION
The command is executed with the given arguments; after it is complete, time reports on standard error the program’s elapsed user time, system time, and real time, in seconds, followed by the command line.

SOURCE
/sys/src/cmd/time.c

SEE ALSO
prof(1)
NAME
    touch - set modification date of a file

SYNOPSIS
    touch [−c ] [−t time ] file ...

DESCRIPTION
    Touch attempts to set the modification time of the files to time (by default, the current time). If a
    file does not exist, it will be created unless option −c is present.

SOURCE
    /sys/src/cmd/touch.c

SEE ALSO
    ls(1), stat(2), chmod(1)

BUGS
    Touch will not touch directories.
NAME
   tr - translate characters

SYNOPSIS
   tr [-c_ds] [ string1 [ string2 ]]

DESCRIPTION
   Tr copies the standard input to the standard output with substitution or deletion of selected characters (runes). Input characters found in string1 are mapped into the corresponding characters of string2. When string2 is short it is padded to the length of string1 by duplicating its last character. Any combination of the options -c_ds may be used:
   -c    Complement string1: replace it with a lexicographically ordered list of all other characters.
   -d    Delete from input all characters in string1.
   -s    Squeeze repeated output characters that occur in string2 to single characters.

In either string a noninitial sequence -x, where x is any character (possibly quoted), stands for a range of characters: a possibly empty sequence of codes running from the successor of the previous code up through the code for x. The character \ followed by 1, 2 or 3 octal digits stands for the character whose 16-bit value is given by those digits. The character sequence \x followed by 1, 2, 3, or 4 hexadecimal digits stands for the character whose 16-bit value is given by those digits. A \ followed by any other character stands for that character.

EXAMPLES
   Replace all upper-case ASCII letters by lower-case.
   tr A-Z a-z <mixed >lower

   Create a list of all the words in file1 one per line in file2, where a word is taken to be a maximal string of alphabetic characters. String2 is given as a quoted newline.
   tr -cs A-Za-z ' '
      ' <file1 >file2

SOURCE
   /sys/src/cmd/tr.c

SEE ALSO
   sed(1)
NAME
trace – show (real-time) process behavior

SYNOPSIS
trace [-d file] [-v] [-w] [pid ...]

DESCRIPTION
Trace displays the behavior of processes running on the machine. In its window it shows a time line for each traced process. Running processes appear as colored blocks, with arrows marking important events in real-time processes (see proc(3)). Black up arrows mark process releases, black down arrows mark process deadlines, green down arrows mark times when a process yielded the processor before its deadline, red down arrows mark times when the process overran its allotted time.

Trace reads /procTRACE to retrieve trace events from the kernel scheduler. Trace events are binary data structures generated by the kernel scheduler. It is assumed that the reader of /procTRACE and the kernel providing it have the same byte order.

The options are:
-d specify an alternate trace event file
-v print events as they are read from the trace event file
-w run in a new window rather than using the current one

Trace recognizes these keystroke commands while it is running:
+ zoom in by a factor of two
- zoom out by a factor of two
p pause or resume
q quit

SEE ALSO
proc(3)

FILES
/procTRACE trace event file
/sys/include/trace.h trace event data structures

SOURCE
/sys/src/cmd/trace.c
NAME
troff, nroff, dpost - text formatting and typesetting

SYNOPSIS
troff [ option ...][ file ...]
dpost [-f][ file ...]
nroff [ option ...][ file ...]

DESCRIPTION
Troff formats text in the named files for printing on a typesetter, emitting a textual intermediate format called ‘typesetter-independent troff output’, understood by programs such as proof(1) and lp(1), but also by a troff post-processor named dpost, which emits corresponding Postscript. Under -f, dpost also emits Postscript font definitions as needed. Nroff does the same as troff, but produces output suitable for typewriter-like devices, usually without further post-processing, but see col(1).

If no file argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input. The options are:

-o list Print pages in the comma-separated list of numbers and ranges. A range N-M means N through M; initial -M means up to M; final N- means from N to the end.
-nN Number first generated page N.
-m name Process the macro file /sys/lib/tmac/tmac.name before the input files.
-raN Set register a (one character name) to N.
-i Read standard input after the input files are exhausted.
-q Invoke the simultaneous input–output mode of the rd request.
-N Produce output suitable for typewriter-like devices.

Typesetter devices (not -N only)
-a Send a printable textual approximation of the results to the standard output.
-T dest Prepare output for typesetter dest:
-Tutf (The default) PostScript printers with preprocessing to handle Unicode characters encoded in UTF
-Tpost Regular PostScript printers
-T202 Mergenthaler Linotron 202
-E dir Take font information from directory dir.

Typewriter (-N) output only
-sN Halt prior to every N pages (default N=1) to allow paper loading or changing.
-T name Prepare output for specified terminal. Known names include utf for the normal Plan 9 UTF encoding of the Unicode Standard character set (default), 37 for the Teletype model 37, lp (line–printer) for any terminal without half–line capability, 450 for the DASi–450 (Diablo Hyterm), and think (HP ThinkJet).
-e Produce equally-spaced words in adjusted lines, using full terminal resolution.
-h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

FILES
/tmp/trtmp* temporary file
/sys/lib/tmac/tmac.* standard macro files
/sys/lib/troff/term/* terminal driving tables for nroff
/sys/lib/troff/font/* font width tables for troff

SOURCE
/sys/src/cmd/troff
/rc/bin/dpost
SEE ALSO

lp(1), proof(1), page(1), eqn(1), tbl(1), pic(1), grap(1), doctype(1), ms(6), image(6), tex(1), deroff(1), col(1)
B. W. Kernighan, “A Typesetter–Independent TROFF”, CSTR #97
NAME
troff2html - convert troff output into HTML

SYNOPSIS
troff2html [-t title] [ file ... ]

DESCRIPTION
Troff2html reads the troff(1) output in the named files, default standard input, and converts them into HTML.

Troff2html does a tolerable job with straight troff output, but it is helped by annotations, described below. Its main use is for man2html (see httpd(8)), which converts man(1) pages into HTML and depends on a specially annotated set of man(6) macros, invoked by troff -manhtml.

Troff output lines beginning

x X html ...

which are introduced by placing \X'html ...' in the input, cause the rest of the line to be interpolated into the HTML produced. Several such lines are recognized specially by troff2html. The most important are the pair

x X html manref start cp 1
x X html manref end cp 1

which are used to create HTML hyperlinks around text of the form cp(1) pointing to /magic/man2html/1/cp.

Troff2html is new and experimental; in time, it may improve and subsume ms2html(1). On the one hand, because it uses the input, ms2html can handle pic(1), eqn(1), etc., which troff2html does not handle at all; on the other hand, ms2html understands only ms(6) documents and is easily confused by complex troff constructions. Troff2html has the reverse properties: it does not handle the preprocessors but its output is reliable and (modulo helper annotations) independent of macro package.

SOURCE
/sys/src/cmd/troff2html

SEE ALSO
troff(1), ms2html(1), man2html in httpd(8).

BUGS
Troff and HTML have different models, and they don’t mesh well in all cases. Troff’s indented paragraphs are not well served in HTML, and the output of troff2html shows this.
NAME
tweak – edit image files, subfont files, face files, etc.

SYNOPSIS
tweak [ file ... ]

DESCRIPTION
Tweak edits existing files holding various forms of images. To create original images, start from an existing image, subfont, etc.

Tweak reads its argument files and displays the resulting images in a vertical column. If the image is too wide to fit across the display, it is folded much like a long line of text in an rio window. Under each image is displayed one or two lines of text presenting its parameters. The first line shows the image's depth, the number of bits per pixel; r, the rectangle covered by the image; and the name of the file from which it was read. If the file is a subfont, a second line presents a hexadecimal 16-bit offset to be applied to character values from the subfont (typically as stored in a font file; see font(6)); and the subfont's n, height, and ascent as defined in cachechars(2).

By means described below, magnified views of portions of the images may be displayed. The text associated with such a view includes mag, the magnification. If the view is of a single character from a subfont, the second line of text shows the character's value (including the subfont's offset) in hexadecimal and as a character in tweak's default font; the character's x, top, bottom, left, and width as defined in cachechars(2); and iwidth, the physical width of the image in the subfont's image.

There are two methods to obtain a magnified view of a character from a subfont. The first is to click mouse button 1 over the image of the character in the subfont. The second is to select the char entry on the button 3 menu, point the resulting gunsight cursor at the desired subfont and click button 3, and then type at the text prompt at the bottom of the screen the character value, either as a multi-digit hexadecimal number or as a single rune representing the character.

To magnify a portion of other types of image files, click button 1 over the unmagnified file. The cursor will switch to a cross. Still with button 1, sweep a rectangle, as in rio, that encloses the portion of the image to be magnified. (If the file is 16×16 or smaller, tweak will just magnify the entire file; no sweeping is necessary.)

Pressing buttons 1 and 2 within magnified images changes pixel values. By default, button 1 sets the pixel to all zeros and button 2 sets the pixel to all ones.

Across the top of the screen is a textual display of global parameters. These values, as well as many of the textual values associated with the images, may be edited by clicking button 1 on the displayed value and typing a new value. The values along the top of the screen are:

mag  Default magnification.
val(hex)
   The value used to modify pixels within magnified images. The value must be in hexadecimal, optionally preceded by a tilde for bitwise negation.
but1
but2  The pixel value written when the corresponding button is pressed over a pixel.
invert–on–copy
   Whether the pixel values are inverted when a copy operation is performed.

Under button 3 is a menu holding a variety of functions. Many of these functions prompt for the image upon which to act by switching to a gunsight cursor; click button 3 over the selection, or click a different button to cancel the action.

open  Read and display a file. The name of the file is typed to the prompt on the bottom line.
read  Reread a file.
write
   Write a file.
copy Use the copy function, default S, to transfer a rectangle of pixels from one image to another. The program prompts with a cross cursor; sweep out a rectangle in one image or just click button 3 to select the whole image. The program will leave that rectangle in place and attach another one to the cursor. Move that rectangle to the desired place in any image and click button 3, or another button to cancel the action.

char As described above, open a magnified view of a character image in a subfont.

pixels Report the coordinate and value of individual pixels indicated by pressing button 3. This is a mode of operation canceled by pressing button 1 or 2.

close Close the specified image. If the image is the unmagnified file, also close any magnified views of that file.

exit Quit tweak. The program will complain once about modified but unwritten files.

SOURCE
/sys/src/cmd/tweak.c

SEE ALSO
  cachechars(2), image(6), font(6)

BUGS
  For a program written to adjust width tables in fonts, tweak has been pushed unreasonably far.
NAME
uniq – report repeated lines in a file

SYNOPSIS
uniq [−udc [−num]][ file]

DESCRIPTION
Uniq copies the input file, or the standard input, to the standard output, comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed. Repeated lines must be adjacent in order to be found.
−u Print unique lines.
−d Print (one copy of) duplicated lines.
−c Prefix a repetition count and a tab to each output line. Implies –u and –d.
−num The first num fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.
+num The first num characters are ignored. Fields are skipped before characters.

SOURCE
/sys/src/cmd/uniq.c

SEE ALSO
sort(1)

BUGS
Field selection and comparison should be compatible with sort(1).
NAME
  units - conversion program

SYNOPSIS
  units [-v][file]

DESCRIPTION
  Units converts quantities expressed in various standard scales to their equivalents in other scales.
  It works interactively in this fashion:

    you have: inch
    you want: cm
      * 2.54
    / 0.393701

  A quantity is specified as a multiplicative combination of units and floating point numbers. Operators have the following precedence:

    + -       add and subtract
    * / × ÷   multiply and divide
    catenation multiply
    ^ \ ^     exponentiation
    |         divide
    ( ... )   grouping

  Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

    pi, π  ratio of circumference to diameter
    c      speed of light
    e      charge on an electron
    g      acceleration of gravity
    force  same as g
    mole   Avogadro's number
    water  pressure head per unit height of water
    au     astronomical unit

  The pound is a unit of mass. Compound names are run together, e.g. lightyear. British units that differ from their US counterparts are prefixed thus: brgallon. Currency is denoted belgiumfranc, britainpound, etc.

  The complete list of units can be found in /lib/units. A file argument to units specifies a file to be used instead of /lib/units. The -v flag causes units to print its entire database.

EXAMPLE
  you have: 15 pounds force/in²
  you want: atm
      * 1.02069
    / .97973

FILES
  /lib/units

SOURCE
  /sys/src/cmd/units.y

BUGS
  Since units does only multiplicative scale changes, it can convert Kelvin to Rankine but not Centigrade to Fahrenheit.
  Currency conversions are only as accurate as the last time someone updated the database.
NAME
upTime – show how long the system has been running

SYNOPSIS
upTime

DESCRIPTION
upTime shows how long the system has been running. It uses the following format:

```
sysname up 33 days, 17:56:42
```

The time given accounts for the timezone.

SOURCE
```
/rc/bin/upTime
```

SEE ALSO
date(1)
NAME
uuencode, uudecode – encode/decode a file as printable ASCII

SYNOPSIS
uuencode [ input ]
uudecode [ input ]

DESCRIPTION
Uuencode and uudecode are filters used to transmit files over transmission media that do not support other than simple ASCII data.
Uuencode converts a file to a purely ASCII–based representation.
Uudecode reads a file produced by uuencode, ignoring any leading and trailing lines that are not part of the encoding, and emits the original file on standard output, also writing its name to standard error.

EXAMPLES
Encode a dis file limbo.dis so that it can be included in a mail message:
   uuencode limbo.dis >tmp
   place tmp in mail message and send to recipient

Decode the mail message ( msg say):
   uudecode <msg >limbo.dis

SOURCE
/sys/src/cmd/uuencode.c
/sys/src/cmd/uudecode.c

SEE ALSO
marshal(1)

BUGS
The encoded file is expanded by at least a third.
This encoding is a relic of the days before MIME encoding.
NAME
vac, unvac - create, extract a vac archive on Venti

SYNOPSIS
vac [-mqsv] [-b blocksize] [-d oldvacfile] [-e exclude] [-f vacfile] [-i name] [-h host] file ...
unvac [-Tctv] [-h host] vacfile [file ...]

DESCRIPTION
Vac creates an archival copy of Plan 9 file trees on Venti. It can be used to build a simple backup
system. One of the unusual properties of Venti is that duplicate blocks are detected and coalesced.
When vac is used on a file tree that shares data with an existing archive, the consumption of stor-
age will be approximately equal to an incremental backup. This reduction in storage consumption
occurs transparently to the user.

As an optimization, the -d and -q options, described below, can be used to explicitly create an
archive relative to an existing archive. These options do not change the resulting archive gener-
ated by vac, but simply reduce the number of write operations to Venti.

The output of vac is the hexadecimal representation of the SHA1 fingerprint of the root of the
archive, in this format:
vac:64daefaec4df4b5cb48a368b361ef56012a4f46

The options to vac are:

-b blocksize Specifies the block size that data will be broken into. The units for the size can be
specified by appending k to indicate kilobytes. The default is 8k. The size must be
in the range of 512 bytes to 52k.

-d oldvacfile Reduce the number of blocks written to Venti by comparing the files to be stored
with the contents of an existing vac file tree whose score is stored in oldvacfile.

-e exclude Do not include the file or directory specified by exclude. This option may be
repeated multiple times.

-f vacfile The results of vac are placed in vacfile, or the standard output if no file is given.

-i name Include standard input as one of the input files, storing it in the archive with the
specified name.

-h host The network address of the Venti server. The default is taken from the environment
variable venti. If this variable does not exist, then the default is the metaname
$venti, which can be configured via ndb(6).

-m Expand and merge any vac archives that are found while reading the input files.
This option is useful for building an archive from a collection of existing archives.
Each archive is inserted into the new archive as if it had been unpacked in the director-
y in which it was found. Multiple archives can be unpacked in a single directory
and the contents will be merged. To be detected, the archives must end in .vac.
Note, an archive is inserted by simply copying the root fingerprint and does not
require the archive to be unpacked.

-q Increase the performance of the -d option by detecting unchanged files based on a
match of the files name and other meta data, rather than examining the contents of
the files.

-s Print out various statistics on standard error.

-v Produce more verbose output on standard error, including the name of the files
added to the archive and the vac archives that are expanded and merged.

Unvac lists or extracts files stored in the vac archive vacfile, which can be either a vac archive
string in the format given above or the name of a file containing one. If file arguments are given,
only those files or directories will be extracted. The options are:

-T Set the modification time on extracted files to the time listed in the archive.
-c  Write extracted files to standard output instead of creating a file.
-h  as per vac.
-t  Print a list of the files to standard output rather than extracting them.
-v  If extracting files, print the name of each file and directory to standard error. If listing files, print metadata in addition to the names.

SOURCE
/sys/src/cmd/vac
/sys/src/cmd/unvac

SEE ALSO
vacfs(4), venti(8)
NAME
read, write, copy - simple Venti clients

SYNOPSIS
venti/read [-h host] [-t type] score
venti/write [-z] [-h host] [-t type]
venti/copy [-fir] [-t type] srchost dsthost score [ type ]

DESCRIPTION
Venti is a SHA1–addressed block storage server. See venti(6) for a full introduction.

Read reads a block with the given score and numeric type from the server host and prints the block
to standard output. If the -h option is omitted, read consults the environment variable $venti
for the name of the Venti server. If the -t option is omitted, read will try each type, one at a time,
until it finds one that works. It prints the corresponding read -t command to standard error to
indicate the type of the block.

Write writes at most 56 kilobytes of data from standard input to the server host and prints the
resulting score to standard output. If the -t option is omitted, write uses type 0, denoting a data
block. If the -z option is given, write zero truncates the block before writing it to the server.

Copy expects score to be the score of a VtRoot block. It copies the entire tree of blocks reachable
from the root block from the server srchost to the server dsthost.

The -f option causes copy to run in ‘fast’ mode, assuming that if a block already exists on the
destination Venti server, all its children also exist and need not be checked.

The -i and -r options control copy’s reaction to errors reading from srchost. Copy always prints
information to standard error about each read error. By default, copy exits after printing the first
error. If the -i option is given, read errors are ignored. This is dangerous behavior because it
breaks the assumption made by ‘fast’ mode. If the -r option is given, copy replaces pointers to
unreadable blocks with pointers to the zero block. It writes the new root score to standard output.

SOURCE
/sys/src/cmd/venti

SEE ALSO
vac(1), venti(2), vacfs(4), venti(6), venti, venti–backup(8), venti–fmt(8)

BUGS
There should be programs to read and write venti files and directories.
NAME
5i, ki, vi, qi – instruction simulators

SYNOPSIS
vi [ textfile ]
vi pid
5i [ textfile ]
5i pid
ki [ textfile ]
ki pid
qi [ textfile ]
qi pid

DESCRIPTION
Vi simulates the execution of a MIPS binary in a Plan 9 environment. It has two main uses: as a debugger and as a statistics gatherer. Programs running under vi execute about two hundred times slower than normal—but faster than single stepping under db. 5i, ki, and qi are similar to vi but interpret ARM, SPARC, and PowerPC binaries. The following discussion refers to vi but applies to the others as well.

Vi will simulate the execution of a named textfile. It will also make a copy of an existing process with process id pid and simulate its continuation.

As a debugger vi offers more complete information than db(1). Tracing can be performed at the level of instructions, system calls, or function calls. Vi allows breakpoints to be triggered when specified addresses in memory are accessed. A report of instruction counts, load delay fills and distribution is produced for each run. Vi simulates the CPU’s caches and MMU to assist the optimization of compilers and programs.

The command interface mirrors the interface to db; see db(1) for a detailed description. Data formats and addressing are compatible with db except for disassembly: vi offers only MIPS (db −mmipsco) mnemonics for machine instructions. Ki offers both Plan 9 and Sun SPARC formats.

Several extra commands allow extended tracing and printing of statistics:

$t[0ics]
The t command controls tracing. Zero cancels all tracing options.
i  Enable instruction tracing
c  Enable call tracing
s  Enable system call tracing

$[itsp]
The i command prints statistics accumulated by all code run in this session.
i  Print instruction counts and frequency.
p  Print cycle profile.
t  (Vi only) Print TLB and cache statistics.
s  Print memory reference, working set and size statistics.

:b[arwe]
Vi allows breakpoints to be set on any memory location. These breakpoints monitor when a location is accessed, read, written, or equals a certain value. For equality the compared value is the count (see db(1)) supplied to the command.

SOURCE
/sys/src/cmd/vi etc.

SEE ALSO
nm(1), db(1)

BUGS
The code generated by the compilers is well supported, but some unusual instructions are unimplemented. Some Plan 9 system calls such as rfork cause simulated traps. The floating point
simulation makes assumptions about the interpreting machine’s floating point support. The floating point conversions performed by \textit{vi} may cause a loss of precision.
NAME
  vncs, vncv - remote frame buffer server and viewer for Virtual Network Computing (VNC)

SYNOPSIS
  vncs [-v] [-c cert] [-d : display] [-g widthxheight] [-p pixfmt] [-x net] [cmd [args]]
  vncs -k : display [-x net]
  vnc [-cstv] [-e encodings] [-k keypattern] host[:n]

DESCRIPTION
  VNC is a lightweight protocol for accessing graphical applications remotely. The protocol allows
  one or more clients to connect to a server. While connected, clients display the frame buffer
  presented by the server and can send mouse events, keyboard events, and exchange snarf buffers.
  The server persists across viewer sessions, so that the virtual application can be accessed from
  various locations as its owner moves around.

  VNC displays have names of the form host : n, where host is the machine’s network name and n is
  a small integer identifier; display n is served on TCP port 5900 + n.

  Vnscs starts a new virtual frame buffer in memory, simulating a Plan 9 terminal running cmd args,
  by default an interactive shell. As viewers connect, each is authenticated using a (rather breakable)
  challenge–response protocol using the user’s Inferno/POP password.

  The options are:

  -c cert      start TLS on each viewer connection using the certificate in the file cert. The corre-
                sponding private key must be loaded into the server’s factotum(4). When serving TLS
                connections, the base port is 35729 rather than 5900.

  -d : n        run on display n ; without this option, the server searches for an unused display.

  -g widthxheight
                set the virtual frame buffer to be widthxheight (default 1024x768) pixels.

  -p pixfmt     set the virtual frame buffer’s internal pixel format to pixfmt (default r5g6b5).

  -v            print verbose output to standard error.

  -x net        announce on an alternate network interface. Because of the weak authentication pro-
                tocol and default lack of encryption, this option must be accompanied by -c.

  The command vnscs -k : n kills the VNC server running on display n.

  Vnc provides access to remote display host : n. It resizes its window to be the smaller of the
  remote frame buffer size and the local screen.

  The options are:

  -c            when connecting to 8-bit displays, request r4g4b4 pixels rather than r3g3b2 pixels.
                This takes up more bandwidth but usually gives significantly better matching to the Plan 9
                color map.

  -e encodings  set the ordered list of allowed frame buffer update encodings. The default (and full)
                set is copyrect corre hextile rre raw. The encodings should be given as a single
                space-separated argument (quoted when using the shell).

  -k keypattern
                add keypattern to the pattern used to select a key from factotum(4).

  -s            share the display with extant viewers; by default extant viewers are closed when a new
                viewer connects.

  -t            start TLS on the connection.

  -v            print verbose output to standard error.

  The VNC protocol represents keyboard input as key up/down events. Plan 9 does not expose the
  state of the Ctl and Shift keys except as it can be inferred from receipt of control or shifted char-
  acters. It does not expose the state of the Alt key at all, since the Alt key is used to compose
Unicode characters (see `keyboard(6)`). Vncv correctly handles the sending of control and shifted characters. To support systems that use key sequences like Alt-X (or worse, Alt-mouse-click), typing the Plan 9 compose sequences Alt Z A (for Alt), Alt Z C (for Ctrl), and Alt Z S (for Shift) will send a “key down” message for the given key. A corresponding “key up” message will be sent after the next key is pressed, or when the sequence is retyped, whichever happens first.

**SOURCE**
/sys/src/cmd/vnc

**SEE ALSO**
`drawterm(8)`
http://www.uk.research.att.com/vnc

**BUGS**
If the remote frame buffer is larger than the local screen, only the upper left corner can be accessed.

Vncv does no verification of the TLS certificate presented by the server.

Vncv supports only version 3.3 of the RFB protocol.
NAME
vt - emulate a VT-100 or VT-220 terminal

SYNOPSIS
vt [-2abcx][-f font][-l log]

DESCRIPTION
Vt replaces a rio window with a fresh instance of the shell, rc(1), running within an emulation of a DEC VT-100 terminal. To exit vt, exit the rc it starts.

Options
2
 a
 x change vt to emulate a VT-220, ANSI, or XTerm terminal respectively.
 b changes the color scheme to white text on a black background, but potentially with colors from escape sequences.
 f changes the color scheme to monochrome (no colors).
 l sets the font.

Menus
The right button has a menu with the following entries to provide the sort of character processing expected by non–Plan 9 systems:

24x80  Resize the vt window to hold 24 rows of 80 columns.
 crrnl Print a newline (linefeed) character after receiving a carriage return from the host.
 cr  Do not print a newline after carriage return.
 nlcr Print a carriage return after receiving a newline from the host.
 nl  Do not print a carriage return after newline.
 raw  Enter raw (no echo, no interpretation) character mode for input.
 cooked Leave raw mode.
 exit  Exit vt.

The middle button has a menu with the following entries:

backup  Move the display back one screenful.
 forward  Move the display forward one screenful. (These are a poor substitute for a scroll bar.)
 reset  Display the last screenful; the same as going forward to the end.
 clear  Clear the screen. Previous contents can be recovered using backup.
 send  Send the contents of the rio snarf buffer, just as send in the rio menu.
 scroll  Make new lines visible as they appear at the bottom.
 page  When the page fills, pause and wait for a character to be typed before proceeding.
        The down arrow key advances a page without sending the character to the host.

SOURCE
/sys/src/cmd/vt

BUGS
This program is used only for communicating with foreign systems, so it is not as rich an emula-
tion as its equivalent in other environments.

Use care in setting raw and newline modes when connecting to Unix systems via con(1) or ssh(1).
It may also be necessary to set the emulator into raw mode.
NAME
   wc - word count

SYNOPSIS
   wc [−lwrbc][ file ... ]

DESCRIPTION
   Wc counts lines, words, runes, syntactically-invalid UTF codes and bytes in the named files, or in the standard input if no file is named. A word is a maximal string of characters delimited by spaces, tabs or newlines. The count of runes includes invalid codes.

   If the optional argument is present, just the specified counts (lines, words, runes, broken UTF codes or bytes) are selected by the letters l, w, r, b, or c. Otherwise, lines, words and bytes (−lwc) are reported.

SOURCE
   /sys/src/cmd/wc.c

BUGS
   The Unicode Standard has many blank characters scattered through it, but wc looks for only ASCII space, tab and newline.

   Wc should have options to count suboptimal UTF codes and bytes that cannot occur in any UTF code.
NAME
weather - print weather report

SYNOPSIS
weather [air] [st]

DESCRIPTION
Weather prints the local conditions and seven–day forecast most recently reported at the US airport with the three–letter location identifier air. Given a two–letter US state abbreviation st instead, weather prints a table of air location identifiers known for st.

The arguments are mutually exclusive and case–insensitive. If neither is given, air defaults to location identifier ewr, designating the Newark, NJ, airport near Bell Labs, Murray Hill.

SOURCE
/rc/bin/weather
NAME
  who, whois - who is using the machine
SYNOPSIS
  who
  whois person
DESCRIPTION
  Who prints the name of everyone with a non-Exiting process on the current machine.
  Whois looks in /adm/whois and /adm/users to find out more information about person.
SOURCE
  /rc/bin/who
NAME
  winwatch - monitor rio windows

SYNOPSIS
  winwatch [−e exclude] [−f font]

DESCRIPTION
  Winwatch displays the labels of all current rio(4) windows, refreshing the display every five seconds. Right clicking a window's label unhides, raises and gives focus to that window. Typing q or DEL quits winwatch.

  If the −e flag is given, windows matching the regular expression exclude are not shown.

EXAMPLE
  Excluding winwatch, stats and faces from being showed.
  % winwatch −e '^(winwatch|stats|faces)'

FILES
  /dev/wsys/*/label

SOURCE
  /sys/src/cmd/winwatch.c

SEE ALSO
  rio(1), rio(4), regexp(6).
NAME
    xd – hex, octal, decimal, or ASCII dump

SYNOPSIS
    xd [ option ... ] [ –format ... ] [ file ... ]

DESCRIPTION
    Xd concatenates and dumps the files (standard input by default) in one or more formats. Groups
    of 16 bytes are printed in each of the named formats, one format per line. Each line of output is
    prefixed by its address (byte offset) in the input file. The first line of output for each group is
    zero–padded; subsequent are blank–padded.

    Formats other than –c are specified by pairs of characters telling size and style, 4x by default. The
    sizes are
    1 or b 1–byte units.
    2 or w 2–byte big–endian units.
    4 or l 4–byte big–endian units.
    8 or v 8–byte big–endian units.

    The styles are
    o    Octal.
    x    Hexadecimal.
    d    Decimal.

    Other options are
    –c    Format as 1x but print ASCII representations or C escape sequences where possible.
    –a style    Print file addresses in the given style (and size 4).
    –u    (Unbuffered) Flush the output buffer after each 16–byte sequence.
    –s    Reverse (swab) the order of bytes in each group of 4 before printing.
    –r    Print repeating groups of identical 16–byte sequences as the first group followed by an
          asterisk.

SOURCE
    /sys/src/cmd/xd.c

SEE ALSO
    db(1)

BUGS
    The various output formats don’t line up properly in the output of xd.
NAME
yacc - yet another compiler-compiler

SYNOPSIS
yacc [ option ... ] grammar

DESCRIPTION
Yacc converts a context-free grammar and translation code into a set of tables for an LR(1) parser
and translator. The grammar may be ambiguous; specified precedence rules are used to break
ambiguities.

The output file, y.tab.c, must be compiled by the C compiler to produce a program yyparse.
This program must be loaded with a lexical analyzer function, yylex(void) (often generated by
lex(1)), with a main(int argc, char *argv[]) program, and with an error handling rou-
tine, yyerror(char*).

The options are
-o output  Direct output to the specified file instead of y.tab.c.
-Dn      Create file y.debug, containing diagnostic messages. To incorporate them in the
          parser, compile it with preprocessor symbol yydebug defined. The amount of
diagnostic output from the parser is regulated by value n. The value 0 reports errors;
1 reports reductions; higher values (up to 4) include more information about state
transitions.
-v      Create file y.output, containing a description of the parsing tables and of con-
flicts arising from ambiguities in the grammar.
-d      Create file y.tab.h, containing #define statements that associate yacc-assigned
‘token codes’ with user-declared ‘token names’. Include it in source files other than
y.tab.c to give access to the token codes.
-s stem  Change the prefix y of the file names y.tab.c, y.tab.h, y.debug, and
          y.output to stem.
-S      Write a parser that uses Stdio instead of the print routines in libc.

The specification of yacc itself is essentially the same as the UNIX version described in the refer-
ences mentioned below. Besides the −D option, the main relevant differences are:

The interface to the C environment is by default through <libc.h> rather than
<stdio.h>; the −S option reverses this.

The parser accepts UTF input text (see utf(6)), which has a couple of effects. First, the
return value of yylex() no longer fits in a short; second, the starting value for non-
terminals is now 0x4000 rather than 257.

The generated parser can be recursive: actions can call yyparse, for example to implement
a sort of #include statement in an interpreter.

Finally, some undocumented inner workings of the parser have been changed, which may
affect programs that know too much about its structure.

FILES
y.output
y.tab.c
y.tab.h
y.debug
y.tmp.*  temporary file
y.acts.*  temporary file
/sys/lib/yaccpar parser prototype
/sys/lib/yaccpars parser prototype using stdio

SOURCE
/sys/src/cmd/yacc.c

SEE ALSO
lex(1)

**BUGS**

The parser may not have full information when it writes to `y.debug` so that the names of the tokens returned by `yy.lex` may be missing.
NAME
yesterday, diffy - print file names from the dump

SYNOPSIS
yesterday [-abcCdDs] [-n daysago] [-date] files ...
diffy [-abcefmnrw] files ...

DESCRIPTION
Yesterday prints the names of the files from the most recent dump. Since dumps are done early in the morning, yesterday's files are really in today's dump. For example, if today is March 17, 1992,
yesterday /adm/users
prints
/n/dump/1992/0317/adm/users

In fact, the implementation is to select the most recent dump in the current year, so the dump selected may not be from today.

When presented with a path of the form /n/fs/path, yesterday will look for dump files of the form /n/fsdump/yyyy/hhmm/path.

By default, yesterday prints the names of the dump files corresponding to the named files. The first set of options changes this behavior.

-a Run acme(1)'s adiff to compare the dump files with the named files.
-b Bind the dump files over the named files.
-c Copy the dump files over the named files.
-C Copy the dump files over the named files only when they differ.
-d Run diff to compare the dump files with the named files.
-D Run diff -n to compare the dump files with the named files.

The date option selects other day's dumps, with a format of 1, 2, 4, 6, or 8 digits of the form d, dd, mddd, yyyyMMdd, or yyyymmmdd.

The -n option selects the dump daysago prior to the current day.

The -s option selects the most recent snapshot instead of the most recent archived dump. Snapshots may occur more frequently than dumps.

Yesterday does not guarantee that the string it prints represents an existing file.

Diffy runs diff(1) with the given options to compare yesterday's version of each of the named files with today's.

EXAMPLES
Back up to yesterday's MIPS binary of vc:
yesterday -c /mips/bin/vc

Temporarily back up to March 1's MIPS C library to see if a program runs correctly when loaded with it:
yesterday -b -0301 /mips/lib/libc.a
rm v.out
mk
v.out

Find what has changed in the C library since March 1:
yesterday -d -0301 /sys/src/libc/port/*.c

Find what has changed in the source tree today:
diffy -r /sys/src

FILES
/n/dump
SOURCE
   /rc/bin/yesterday
   /rc/bin/diffy

SEE ALSO
   *history*(1), *bind*(1), *diff*(1), *fs*(4).

BUGS
   It’s hard to use this command without singing.
NAME
intro - introduction to library functions

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <auth.h>
#include <bio.h>
#include <draw.h>
#include <fcall.h>
#include <frame.h>
#include <mach.h>
#include <ndb.h>
#include <regexp.h>
#include <stdio.h>
#include <thread.h>

DESCRIPTION
This section describes functions in various libraries. For the most part, each library is defined by a single C include file, such as those listed above, and a single archive file containing the library proper. The name of the archive is /$objtype/lib/libx.a, where x is the base of the include file name, stripped of a leading lib if present. For example, <draw.h> defines the contents of library /$objtype/lib/libdraw.a, which may be abbreviated when named to the loader as -ldraw. In practice, each include file contains a #pragma that directs the loader to pick up the associated archive automatically, so it is rarely necessary to tell the loader which libraries a program needs.

The library to which a function belongs is defined by the header file that defines its interface. The 'C library', libc, contains most of the basic subroutines such as strlen. Declarations for all of these functions are in <libc.h>, which must be preceded by (needs) an include of <u.h>. The graphics library, draw, is defined by <draw.h>, which needs <libc.h> and <u.h>. The Buffered I/O library, libbio, is defined by <bio.h>, which needs <libc.h> and <u.h>. The ANSI C Standard I/O library, libstdio, is defined by <stdio.h>, which needs <u.h>. There are a few other, less commonly used libraries defined on individual pages of this section.

The include file <u.h>, a prerequisite of several other include files, declares the architecture-dependent and -independent types, including: uchar, ushort, uint, and ulong, the unsigned integer types; schar, the signed char type; vlong and uvlong, the signed and unsigned very long integral types; Rune, the Unicode character type; u8int, u16int, u32int, and u64int, the unsigned integral types with specific widths; uintptr, the unsigned integral type with the same width as a pointer; jmp_buf, the type of the argument to setjmp and longjmp, plus macros that define the layout of jmp_buf (see setjmp(2)); definitions of the bits in the floating-point control register as used by getfcr(2); and the macros va_arg and friends for accessing arguments of variadic functions (identical to the macros defined in <stdarg.h> in ANSI C).

Name space
Files are collected into a hierarchical organization called a file tree starting in a directory called the root. File names, also called paths, consist of a number of /-separated path elements with the slashes corresponding to directories. A path element must contain only printable characters (those outside the control spaces of ASCII and Latin–1). A path element cannot contain a slash.

When a process presents a file name to Plan 9, it is evaluated by the following algorithm. Start with a directory that depends on the first character of the path: / means the root of the main hierarchy, # means the separate root of a kernel device’s file tree (see Section 3), and anything else means the process’s current working directory. Then for each path element, look up the element in the directory, advance to that directory, do a possible translation (see below), and repeat. The last step may yield a directory or regular file. The collection of files reachable from the root is
called the name space of a process.

A program can use bind or mount (see bind(2)) to say that whenever a specified file is reached during evaluation, evaluation instead continues from a second specified file. Also, the same system calls create union directories, which are concatenations of ordinary directories that are searched sequentially until the desired element is found. Using bind and mount to do name space adjustment affects only the current process group (see below). Certain conventions about the layout of the name space should be preserved; see namespace(4).

File I/O

Files are opened for input or output by open or create (see open(2)). These calls return an integer called a file descriptor which identifies the file to subsequent I/O calls, notably read(2) and write. The system allocates the numbers by selecting the lowest unused descriptor. They are allocated dynamically; there is no visible limit to the number of file descriptors a process may have open. They may be reassigned using dup(2). File descriptors are indices into a kernel resident file descriptor table. Each process has an associated file descriptor table. In some cases (see rfork in fork(2)) a file descriptor table may be shared by several processes.

By convention, file descriptor 0 is the standard input, 1 is the standard output, and 2 is the standard error output. With one exception, the operating system is unaware of these conventions; it is permissible to close file 0, or even to replace it by a file open only for writing, but many programs will be confused by such chicanery. The exception is that the system prints messages about broken processes to file descriptor 2.

Files are normally read or written in sequential order. The I/O position in the file is called the file offset and may be set arbitrarily using the seek(2) system call.

Directories may be opened and read much like regular files. They contain an integral number of records, called directory entries. Each entry is a machine-independent representation of the information about an existing file in the directory, including the name, ownership, permission, access dates, and so on. The entry corresponding to an arbitrary file can be retrieved by stat(2) or fstat; wstat and fwstat write back entries, thus changing the properties of a file. An entry may be translated into a more convenient, addressable form called a Dir structure; dirstat, dirfstat, dirwstat, and dirfwstat execute the appropriate translations (see stat(2)).

New files are made with create (see open(2)) and deleted with remove(2). Directories may not directly be written; create, remove, wstat, and fwstat alter them.

The operating system kernel records the file name used to access each open file or directory. If the file is opened by a local name (one that does not begin / or #), the system makes the stored name absolute by prefixing the string associated with the current directory. Similar lexical adjustments are made for path names containing . (dot) or .. (dot-dot). By this process, the system maintains a record of the route by which each file was accessed. Although there is a possibility for error—the name is not maintained after the file is opened, so removals and renamings can confound it—this simple method usually permits the system to return, via the fd2path(2) system call and related calls such as getwd(2), a valid name that may be used to find a file again. This is also the source of the names reported in the name space listing of ns(1) or /dev/ns (see proc(3)).

Pipe(2) creates a connected pair of file descriptors, useful for bidirectional local communication.

Process execution and control

A new process is created when an existing one calls rfork with the RPROC bit set, usually just by calling fork(2). The new (child) process starts out with copies of the address space and most other attributes of the old (parent) process. In particular, the child starts out running the same program as the parent; exec(2) will bring in a different one.

Each process has a unique integer process id; a set of open files, indexed by file descriptor; and a current working directory (changed by chdir(2)).

Each process has a set of attributes — memory, open files, name space, etc. — that may be shared or unique. Flags to rfork control the sharing of these attributes.

The memory of a process is divided into segments. Every program has at least a text (instruction) and stack segment. Most also have an initialized data segment and a segment of zero-filled data called bss. Processes may segattach(2) other segments for special purposes.
A process terminates by calling \texttt{exits(2)}. A parent process may call \texttt{wait(2)} to wait for some child to terminate. A string of status information may be passed from \texttt{exits} to \texttt{wait}. A process can go to sleep for a specified time by calling \texttt{sleep(2)}.

There is a \textit{notification} mechanism for telling a process about events such as address faults, floating point faults, and messages from other processes. A process uses \texttt{notify(2)} to register the function to be called (the \textit{notification handler}) when such events occur.

**Multithreading**

By calling \texttt{rfork} with the RFMEM bit set, a program may create several independently executing processes sharing the same memory (except for the stack segment, which is unique to each process). Where possible according to the ANSI C standard, the main C library works properly in multiprocess programs; \texttt{malloc}, \texttt{printf}, and the other routines use locks (see \texttt{lock(2)}) to synchronize access to their data structures. The graphics library defined in \texttt{<graphics.h>} is also multi-process capable; details are in \textit{graphics(2)}. In general, though, multiprocess programs should use some form of synchronization to protect shared data.

The thread library, defined in \texttt{<thread.h>}, provides support for multiprocess programs. It includes a data structure called a Channel that can be used to send messages between processes, and coroutine–like \texttt{threads}, which enable multiple threads of control within a single process. The threads within a process are scheduled by the library, but there is no pre-emptive scheduling within a process; thread switching occurs only at communication or synchronization points.

Most programs using the thread library comprise multiple processes communicating over channels, and within some processes, multiple threads. Since Plan 9 I/O calls may block, a system call may block all the threads in a process. Therefore, a program that shouldn’t block unexpectedly will use a process to serve the I/O request, passing the result to the main processes over a channel when the request completes. For examples of this design, see \textit{ioproc(2) or mouse(2)}.

SEE ALSO

\texttt{nm(1), 2f(1), 2c(1)}

**DIAGNOSTICS**

Math functions in \textit{libc} return special values when the function is undefined for the given arguments or when the value is not representable (see \texttt{nan(2)}).

Some of the functions in \textit{libc} are system calls and many others employ system calls in their implementation. All system calls return integers, with \texttt{-1} indicating that an error occurred; \texttt{errstr(2)} recovers a string describing the error. Some user–level library functions also use the \texttt{errstr} mechanism to report errors. Functions that may affect the value of the error string are said to “set \texttt{errstr}”; it is understood that the error string is altered only if an error occurs.
NAME

Srv, dirread9p, emalloc9p, erealloc9p, estrdup9p, listensrv, postfd, postmountsrv, readbuf, readstr, respond, responderror, threadlistensrv, threadpostmountsrv, srv – 9P file service

SYNOPSIS

#include <u.h>
#include <libc.h>
#include <fcall.h>
#include <thread.h>
#include <9p.h>

typedef struct Srv {
  Tree* tree;

  void (*attach)(Req *r);
  void (*auth)(Req *r);
  void (*open)(Req *r);
  void (*create)(Req *r);
  void (*read)(Req *r);
  void (*write)(Req *r);
  void (*remove)(Req *r);
  void (*flush)(Req *r);
  void (*stat)(Req *r);
  void (*wstat)(Req *r);
  void (*walk)(Req *r);

  char* (*walk1)(Fid *fid, char *name, Qid *qid);
  char* (*clone)(Fid *oldfid, Fid *newfid);

  void (*destroyfid)(Fid *fid);
  void (*destroyreq)(Req *r);
  void (*end)(Srv *s);
  void* aux;

  int infd;
  int outfd;
  int srvfd;
  int nopipe;
} Srv;

int srv(Srv *s)
void postmountsrv(Srv *s, char *name, char *mtpt, int flag)
void threadpostmountsrv(Srv *s, char *name, char *mtpt, int flag)
void listensrv(Srv s, char *addr)
void threadlistensrv(Srv s, char *addr)
int postfd(char *srvname, int fd)
void respond(Req *r, char *error)
void responderror(Req*)
void readstr(Req *r, char *src)
void readbuf(Req *r, void *src, long nsrс)
typedef int Dirgen(int n, Dir *dir, void *aux)
void dirread9p(Req *r, Dirgen *gen, void *aux)
void walkandclone(Req *r, char *(*walk1)(Fid *old, char *name, void *v),
  char *(*clone)(Fid *old, Fid *new, void *v), void *v)
void* emalloc9p(ulong n)
void* erealloc9p(void *v, ulong n)
char* estrdup9p(char *s)
extern int chatty9p;
DESCRIPTION

The function \texttt{srv} serves a 9P session by reading requests from \texttt{s->infld}, dispatching them to the function pointers kept in \texttt{SrV}, and writing the responses to \texttt{s->outfd}. (Typically, \texttt{postmountsrv} or \texttt{threadpostmountsrv} initializes the \texttt{infld} and \texttt{outfd} structure members. See the description below.)

\texttt{Req} and \texttt{Fid} structures are allocated one-to-one with uncompleted requests and active fids, and are described in \texttt{9pfid(2)}.

The behavior of \texttt{srv} depends on whether there is a file tree (see \texttt{9pfile(2)}) associated with the server, that is, whether the \texttt{tree} element is nonzero. The differences are made explicit in the discussion of the service loop below. The aux element is the client’s, to do with as it pleases.

\texttt{SrV} does not return until the 9P conversation is finished. Since it is usually run in a separate process so that the caller can exit, the service loop has little chance to return gracefully on out of memory errors. It calls \texttt{emalloc9p}, \texttt{erealloc9p}, and \texttt{estrdup9p} to obtain its memory. The default implementations of these functions act as \texttt{malloc}, \texttt{realloc}, and \texttt{strdup} but abort the program if they run out of memory. If alternate behavior is desired, clients can link against alternate implementations of these functions.

\texttt{Postmountsrv} and \texttt{threadpostmountsrv} are wrappers that create a separate process in which to run \texttt{srv}. They do the following:

\begin{itemize}
  \item If \texttt{s->nopipe} is zero (the common case), initialize \texttt{s->infld} and \texttt{s->outfd} to be one end of a freshly allocated pipe, with \texttt{s->srvfd} initialized as the other end.
  \item If \texttt{name} is non-nil, call \texttt{postfd(s->srvfd, name)} to post \texttt{s->srvfd} as \texttt{/srv/name}.
  \item Fork a child process via \texttt{rfork} (see \texttt{fork(2)}) or \texttt{procrfork} (see \texttt{thread(2)}), using the RFFDG, RFNAMEG, and RFMEM flags. The child process calls \texttt{close(s->srvfd)} and then \texttt{srv(s)}; it will exit once \texttt{srv} returns.
  \item If \texttt{mtpt} is non-nil, call \texttt{amount(s->srvfd, mtpt, flag, "")}; otherwise, close \texttt{s->srvfd}.
  \item The parent returns to the caller.
\end{itemize}

If any error occurs during this process, the entire process is terminated by calling \texttt{sysfatal} (see \texttt{perror(2)}).

\texttt{Listensrv} and \texttt{threadlistensrv} create a separate process to announce as \texttt{addr}. The process listens for incoming connections, creating a new process to serve each. Using these functions results in \texttt{srv} and the service functions being run in multiple processes simultaneously. The library locks its own data structures as necessary; the client may need to lock data it shares between the multiple connections.

Service functions

The functions in a \texttt{SrV} structure named after 9P transactions are called to satisfy requests as they arrive. If a function is provided, it must arrange for \texttt{respond} to be called when the request is satisfied. The only parameter of each service function is a \texttt{Req*} parameter (say \texttt{r}). The incoming request parameters are stored in \texttt{r->ifcall}; \texttt{r->fid} and \texttt{r->newfid} are pointers to \texttt{Fid} structures corresponding to the numeric fids in \texttt{r->ifcall}; similarly, \texttt{r->oldreq} is the \texttt{Req} structure corresponding to \texttt{r->ifcall.oldtag}. The outgoing response data should be stored in \texttt{r->ofcall}. The one exception to this rule is that \texttt{stat} should fill in \texttt{r->ofcall.stat} rather than \texttt{r->ofcall.stat}: the library will convert the structure into the machine-independent wire representation. Similarly, \texttt{wstat} may consult \texttt{r->d} rather than decoding \texttt{r->ifcall.stat} itself. When a request has been handled, \texttt{respond} should be called with \texttt{r} and an error string. If the request was satisfied successfully, the error string should be a nil pointer. Note that it is permissible for a function to return without itself calling \texttt{respond}, as long as it has arranged for \texttt{respond} to be called at some point in the future by another proc sharing its address space, but see the discussion of \texttt{flush} below. Once \texttt{respond} has been called, the \texttt{Req*} as well as any pointers it once contained must be considered freed and not referenced.

\texttt{Responderror} calls \texttt{respond} with the system error string (see \texttt{errstr(2)}).

If the service loop detects an error in a request (e.g., an attempt to reuse an extant fid, an open of an already open fid, a read from a fid opened for write, etc.) it will reply with an error without consulting the service functions.
The service loop provided by srv (and indirectly by postmountsrv and threadpostmountsrv) is single-threaded. If it is expected that some requests might block, arranging for alternate processes to handle them is suggested.

The constraints on the service functions are as follows. These constraints are checked while the server executes. If a service function fails to do something it ought to have, srv will call endsrv and then abort.

**Auth** If authentication is desired, the auth function should record that r->afid is the new authentication fid and set r->afid->qid and ofcall.qid. Auth may be nil, in which case it will be treated as having responded with the error “argc0: authentication not required,” where argc0 is the program name variable as set by ARGBEGIN (see argc(2)).

**Attach** The attach function should check the authentication state of afid if desired, and set r->fid->qid and ofcall.qid to the qid of the file system root. Attach may be nil only if file trees are in use; in this case, the qid will be filled from the root of the tree, and no authentication will be done.

**Walk** If file trees are in use, walk is handled internally, and srv->walk is never called.

If file trees are not in use, walk should consult r->ifcall . wname and r->ifcall . nname, filling in ofcall . qid and ofcall . nqid, and also copying any necessary aux state from r->fid to r->newfid when the two are different. As long as walk sets ofcall . nqid appropriately, it can respond with a nil error string even when 9P demands an error (e.g., in the case of a short walk); the library detects error conditions and handles them appropriately.

Because implementing the full walk message is intricate and prone to error, the helper routine walkandclone will handle the request given pointers to two functions walk1 and (optionally) clone . Clone, if non-nil, is called to signal the creation of newfid from oldfid. Typically a clone routine will copy or increment a reference count in oldfid’s aux element.

Walk1 should walk fid to name, initializing fid->qid to the new path’s qid. Both should return nil on success or an error message on error. Walkandclone will call respond after handling the request.

**Walk1, Clone**

If the client provides functions srv->walk1 and (optionally) srv->clone, the 9P service loop will call walkandclone with these functions to handle the request. Unlike the walk1 above, srv->walk1 must fill in both fid->qid and *qid with the new qid on a successful walk.

**Open** If file trees are in use, the file metadata will be consulted on open, create, remove, and wstat to see if the requester has the appropriate permissions. If not, an error will be sent back without consulting a service function.

If not using file trees or the user has the appropriate permissions, open is called with r->ofcall . qid already initialized to the one stored in the Fid structure (that is, the one returned in the previous walk). If the qid changes, both should be updated.

**Create** The create function must fill in both r->fid->qid and r->ofcall . qid on success. When using file trees, create should allocate a new File with createfile; note that createfile may return nil (because, say, the file already exists). If the create function is nil, srv behaves as though it was a function that always responded with the error “create prohibited”.

**Remove** Remove should mark the file as removed, whether by calling removefile when using file trees, or by updating an internal data structure. In general it is not a good idea to clean up the aux information associated with the corresponding File at this time, to avoid memory errors if other fids have references to that file. Instead, it is suggested that remove simply mark the file as removed (so that further operations on it know to fail) and wait until the file tree’s destroy function is called to reclaim the aux pointer. If not using file trees, it is prudent to take the analogous measures. If remove is not provided, all remove requests will draw “remove prohibited” errors.

**Read** The read function must be provided; it fills r->ofcall . data with at most r->ifcall . count bytes of data from offset r->ifcall . offset of the file. It also sets r->ofcall . count to the number of bytes being returned. If using file trees, srv will handle reads of directories internally, only calling read for requests on files. Readstr and readbuf are useful for
satisfying read requests on a string or buffer. Consulting the request in \texttt{r->ifcall}, they fill \texttt{r->ifcall.data} and set \texttt{r->ifcall.count}; they do not call \texttt{respond}. Similarly, \texttt{dirread9p} can be used to handle directory reads in servers not using file trees. The passed \texttt{gen} function will be called as necessary to fill \texttt{dir} with information for the \texttt{nth} entry in the directory. The string pointers placed in \texttt{dir} should be fresh copies made with \texttt{estrapup9p}; they will be freed by \texttt{dirread9p} after each successful call to \texttt{gen}. \texttt{Gen} should return zero if it successfully filled \texttt{dir}, minus one on end of directory.

\textbf{Write} The \texttt{write} function is similar but need not be provided. If it is not, all writes will draw "write prohibited" errors. Otherwise, \texttt{write} should attempt to write the \texttt{r->ifcall.count} bytes of \texttt{r->ifcall.data} to offset \texttt{r->ifcall.offset} of the file, setting \texttt{r->ifcall.count} to the number of bytes actually written. Most programs consider it an error to write less than the requested amount.

\textbf{Stat} \texttt{Stat} should fill \texttt{r->d} with the stat information for \texttt{r->fid}. If using file trees, \texttt{r->d} will have been initialized with the stat info from the tree, and \texttt{stat} itself may be nil.

\textbf{Wstat} The \texttt{wstat} consults \texttt{r->d} in changing the metadata for \texttt{r->fid} as described in \texttt{stat(5)}. When using file trees, \texttt{srv} will take care to check that the request satisfies the permissions outlined in \texttt{stat(5)}. Otherwise \texttt{wstat} should take care to enforce permissions where appropriate.

\textbf{Flush} Servers that always call \texttt{respond} before returning from the service functions need not provide a \texttt{flush} implementation: \texttt{flush} is only necessary in programs that arrange for \texttt{respond} to be called asynchroneously. \texttt{Flush} should cause the request \texttt{r->oldreq} to be cancelled or hurried along. If \texttt{oldreq} is cancelled, this should be signalled by calling \texttt{respond on oldreq} with error string ‘interrupted’. \texttt{Flush} must respond to \texttt{r} with a nil error string. \texttt{Flush} may respond to \texttt{r} before forcing a response to \texttt{r->oldreq}. In this case, the library will delay sending the \texttt{Rflush} message until the response to \texttt{r->oldreq} has been sent.

\textbf{Destroyfid}, \textbf{destroyreq}, and \textbf{end} are auxiliary functions, not called in direct response to 9P requests.

\textbf{Destroyfid} When a \texttt{Fid}'s reference count drops to zero (\textit{i.e.}, it has been clunked and there are no outstanding requests referring to it), \texttt{destroyfid} is called to allow the program to dispose of the \texttt{fid->aux} pointer.

\textbf{Destroyreq} Similarly, when a \texttt{Req}'s reference count drops to zero (\textit{i.e.}, it has been handled via \texttt{respond} and other outstanding pointers to it have been closed), \texttt{destroyreq} is called to allow the program to dispose of the \texttt{r->aux} pointer.

\textbf{End} Once the 9P service loop has finished (end of file have reached on the service pipe or a bad message has been read), \texttt{end} is called (if provided) to allow any final cleanup. For example, it was used by the Palm Pilot synchronization file system (never finished) to gracefully terminate the serial conversation once the file system had been unmouted. After calling \texttt{end}, the service loop (which runs in a separate process from its caller) terminates using _exits (see \texttt{exits(2)}).

If the \texttt{chatty9p} flag is at least one, a transcript of the 9P session is printed on standard error. If the \texttt{chatty9p} flag is greater than one, additional unspecified debugging output is generated. By convention, servers written using this library context the \texttt{--D} option to increment \texttt{chatty9p}.

\textbf{EXAMPLES} \texttt{Archfs(4), cdfs(4), nntpfs(4), snap(4), and /sys/src/lib9p/ramfs.c} are good examples of simple single-threaded file servers. \texttt{Webfs(4)} and \texttt{sshnet} (see \texttt{ssh(1)}) are good examples of multi-threaded file servers.

In general, the \texttt{File} interface is appropriate for maintaining arbitrary file trees (as in \texttt{ramfs}). The \texttt{File} interface is best avoided when the tree structure is easily generated as necessary; this is true when the tree is highly structured (as in \texttt{cdfs} and \texttt{nttpfs}) or is maintained elsewhere.

\textbf{SOURCE} \texttt{/sys/src/lib9p}
SEE ALSO
9pfid(2), 9pfile(2), srv(3), intro(5)

BUGS
The switch to 9P2000 was taken as an opportunity to tidy much of the interface; we promise to avoid such gratuitous change in the future.
NAME
Cmdbuf, parsecmd, respondcmderror, lookupcmd - control message parsing

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <fcall.h>
#include <thread.h>
#include <9p.h>
typedef struct Cmdbuf
{
    char  *buf;
    char  **f;
    int   nf;
} Cmdbuf;

typedef struct Cmdtab
{
    int   index;
    char  *cmd;
    int   narg;
};
Cmdbuf  *parsecmd(char *p, int n)
Cmdtab  *lookupcmd(Cmdbuf *cb, Cmdtab *tab, int ntab)
void     respondcmderror(Req *r, Cmdbuf *cb, char *fmt, ...)

DESCRIPTION
These data structures and functions provide parsing of textual control messages.

Parsecmd treats the n bytes at p (which need not be NUL-terminated) as a UTF string and splits it
using tokenize (see getfields(2)). It returns a Cmdbuf structure holding pointers to each field in
the message. It is the caller’s responsibility to free this structure when it is no longer needed.

Lookupcmd walks through the array ctab, which has ntab entries, looking for the first Cmdtab
that matches the parsed command. (If the parsed command is empty, lookupcmd returns nil
immediately.) A Cmdtab matches the command if cmd is equal to cb->f[0] or if cmd is *. Once a matching
Cmdtab has been found, if narg is not zero, then the parsed command must have exactly
narg fields (including the command string itself). If the command has the wrong
number of arguments, lookupcmd returns nil. Otherwise, it returns a pointer to the Cmdtab
entry. If lookupcmd does not find a matching command at all, it returns nil. Whenever lookupcmd
returns nil, it sets the system error string.

Respondcmderror responds to request r with an error of the form ‘fmt cmd,’ where fmt is the
formatted string and cmd is a reconstruction of the parsed command. Fmt is often simply %r .

EXAMPLES
This interface is not used in any distributed 9P servers. It was lifted from the Plan 9 kernel.
Almost any kernel driver (/sys/src/9/*dev*.c) is a good example.

SOURCE
/sys/src/lib9p/parse.c

SEE ALSO
9p(2)
NAME
   Fid, Fidpool, allocfidpool, freefidpool, allocfid, closefid, lookupfid, removefid, Req, Reqpool, 
   allocreqpool, freereqpool, allocreq, closereq, lookupreq, removereq – 9P fid, request tracking

SYNOPSIS
   #include <u.h>
   #include <libc.h>
   #include <fcall.h>
   #include <thread.h>
   #include <9p.h>

typedef struct Fid {
   ulong fid;
   char omode; /* -1 if not open */
   char *uid;
   Qid qid;
   File *file;
   void *aux;
   ... 
} Fid;

typedef struct Req {
   ulong tag;
   Fcall ifcall;
   Fcall ofcall;
   Req *oldreq;
   void *aux;
   Fid *fid;
   Fid *afid;
   Fid *newfid;
   ...
} Req;

Fidpool* allocfidpool(void (*destroy)(Fid*))
void freefidpool(Fidpool *p)
Fid* allocfid(Fidpool *p, ulong fid)
Fid* lookupfid(Fidpool *p, ulong fid)
Fid* removefid(Fidpool *p, ulong fid);
void closefid(Fid *f)

Reqpool* allocreqpool(void (*destroy)(Req*))
void freereqpool(Reqpool *p)
Req* allocreq(Reqpool *p, ulong tag)
Req* lookupreq(Reqpool *p, ulong tag)
Req* removereq(Reqpool *p, ulong tag);
void closereq(Req *f)

DESCRIPTION
   These routines provide management of Fid and Req structures from Fidpools and Reqpools. 
   They are primarily used by the 9P server loop described in 9p(2).

   Fid structures are intended to represent active fids in a 9P connection, as Chan structures do in 
   the Plan 9 kernel. The fid element is the integer fid used in the 9P connection. Omode is the 
   mode under which the fid was opened, or -1 if this fid has not been opened yet. Note that in 
   addition to the values OREAD, OWRITE, and ORDWR, omode can contain the various flags permissible 
   in an open call. To ignore the flags, use omode&OMASK. Omode should not be changed by 
   the client. The fid derives from a successful authentication by uid. Qid contains the qid 
   returned in the last successful walk or create transaction involving the fid. In a file tree-based 
   server, the Fid’s file element points at a File structure (see 9pfile(2)) corresponding to the 
   fid. The aux member is intended for use by the client to hold information specific to a particular
Fid. With the exception of aux, these elements should be treated as read-only by the client.

Allocfidpool creates a new Fidpool. Freefidpool destroys such a pool. Allocfid returns a new Fid whose fid number is fid. There must not already be an extant Fid with that number in the pool. Once a Fid has been allocated, it can be looked up by fid number using lookupfid. Fids are reference counted: both allocfid and lookupfid increment the reference count on the Fid structure before returning. When a reference to a Fid is no longer needed, closefid should be called to note the destruction of the reference. When the last reference to a Fid is removed, if destroy (supplied when creating the fid pool) is not zero, it is called with the Fid as a parameter. It should perform whatever cleanup is necessary regarding the aux element. Removefid is equivalent to lookupfid but also removes the Fid from the pool. Note that due to lingering references, the return of removefid may not mean that destroy has been called.

Allocreqpool, freereqpool, allocreq, lookupreq, closereq, and removereq are analogous but operate on Req pools and Req structures.

SOURCE
/sys/src/lib9p

SEE ALSO
9p(2), 9pfile(2)
NAME
Tree, alloctree, freetree, File, createfile, closefile, removefile, walkfile, opendirfile, readdirfile, 
closedirfile, hasperm - in-memory file hierarchy

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <fcall.h>
#include <thread.h>
#include <9p.h>
typedef struct File
{
    Ref;
    Dir;
    void*aux;
    ...
} File;
typedef struct Tree
{
    File *root;
    ...
} Tree;
Tree* alloctree(char *uid, char *gid, ulong mode, 
    void (*destroy)(File*))
void freetree(Tree *tree)
File* createfile(File *dir, char *name, char *uid, 
    ulong mode, void *aux)
int removefile(File *file)
void closefile(File *file)
File* walkfile(File *dir, char *path)
Readdir* opendirfile(File *dir)
long readdirfile(Readdir *rdir, uchar *buf, long n)
void closedirfile(Readdir *rdir)
int hasperm(File *file, char *uid, int p)

DESCRIPTION
Files and Trees provide an in-memory file hierarchy intended for use in 9P file servers.

Alloctree creates a new tree of files, and freetree destroys it. The root of the tree (also the root 
element in the structure) will have mode mode and be owned by user uid and group gid. Destroy is 
used when freeing File structures and is described later.

Files (including directories) other than the root are created using createfile, which attempts to 
create a file named name in the directory dir. If created, the file will have owner uid and have a 
group inherited from the directory. Mode and the permissions of dir are used to calculate the per-
mission bits for the file as described in open(5). It is permissible for name to be a slash-separated 
path rather than a single element.

Removefile removes a file from the file tree. The file will not be freed until the last reference to it 
has been removed. Directories may only be removed when empty. Removefile returns zero on 
success, -1 on error. It is correct to consider removefile to be closefile with the side effect of 
removing the file when possible.

Walkfile evaluates path relative to the directory dir, returning the resulting file, or zero if the 
named file or any intermediate element does not exist.

The File structure’s aux pointer may be used by the client for per-File storage. Files are 
reference-counted: if not zero, destroy (specified in the call to alloctree) will be called for each file 
when its last reference is removed or when the user is freed. Destroy should take care of any nec-
necessary cleanup related to aux. When creating new file references by copying pointers, call inc-
ref (see lock(2)) to update the reference count. To note the removal of a reference to a file, call
closefile.  Createfile and walkfile return new references.  Removefile, closefile, and walkfile (but not createfile) consume the passed reference.

Directories may be read, yielding a directory entry structure (see stat(5)) for each file in the directory. In order to allow concurrent reading of directories, clients must obtain a Readdir structure by calling opendirfile on a directory. Subsequent calls to readdirfile will each yield an integral number of machine-independent stat buffers, until end of directory. When finished, call closedirfile to free the Readdir.

Hasperm does simplistic permission checking; it assumes only one-user groups named by uid and returns non-zero if uid has permission p (a bitwise-or of AREAD, AWRITE and AEXEC) according to file->mode. 9P servers written using File trees will do standard permission checks automatically; hasperm may be called explicitly to do additional checks. A 9P server may link against a different hasperm implementation to provide more complex groups.

EXAMPLE
The following code correctly handles references when elementwise walking a path and creating a file.

```c
f = tree->root;
incf(f);
for(i=0; i<n && f!=nil; i++)
    f = walkfile(f, elem[i]);
if(f == nil)
    return nil;
nf = createfile(f, "foo", "nls", 0666, nil);
closefile(f);
return nf;
```

SOURCE
/sys/src/lib9p/file.c

SEE ALSO
9p(2)

BUGS
The reference counting is cumbersome.
NAME
   abort - generate a fault

SYNOPSIS
   #include <u.h>
   #include <libc.h>
   void abort(void)

DESCRIPTION
   **Abort** causes an access fault, causing the current process to enter the ‘Broken’ state. The process can then be inspected by a debugger.

SOURCE
   /sys/src/libc/9sys/abort.c
NAME
  abs, labs - integer absolute values

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  int  abs(int a)
  long labs(long a)

DESCRIPTION
  Abs returns the absolute value of integer \( a \), and \( labs \) does the same for a long.

SOURCE
  /sys/src/libc/port/abs.c

SEE ALSO
  floor(2) for fabs

DIAGNOSTICS
  Abs and \( labs \) return the most negative integer or long when the true result is unrepresentable.
NAME
  access - determine accessibility of file

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  int access(char *name, int mode)

DESCRIPTION
  Access evaluates the given file name for accessibility. If mode&4 is nonzero, read permission is
  expected; if mode&2, write permission; if mode&1, execute permission. If mode==0, the file
  merely need exist. In any case all directories leading to the file must permit searches. Zero is
  returned if the desired access is permitted, -1 if not.

  Only access for open is checked. A file may look executable, but exec(2) will fail unless it is in
  proper format.

  The include file defines AEXIST=0, AEXEC=1, AWRITE=2, and AREAD=4.

SOURCE
  /sys/src/libc/9sys/access.c

SEE ALSO
  stat(2)

DIAGNOSTICS
  Sets errstr.

BUGS
  Since file permissions are checked by the server and group information is not known to the client,
  access must open the file to check permissions. (It calls stat(2) to check simple existence.)
NAME
addpt, subpt, mulpt, divpt, rectaddpt, rectspt, insetrect, canonrect, eqpt, eqrect, ptinrect, rectin-
rect, rectXrect, rectclip, combinerect, Dx, Dy, Pt, Rect, Rpt – arithmetic on points and rectangles

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
Point    addpt(Point p, Point q)
Point    subpt(Point p, Point q)
Point    mulpt(Point p, int a)
Point    divpt(Point p, int a)
Rectangle rectaddpt(Rectangle r, Point p)
Rectangle rectspt(Rectangle r, Point p)
Rectangle insetrect(Rectangle r, int n)
Rectangle canonrect(Rectangle r)
int      eqpt(Point p, Point q)
int      eqrect(Rectangle r, Rectangle s)
int      ptinrect(Point p, Rectangle r)
int      rectinrect(Rectangle r, Rectangle s)
int      rectXrect(Rectangle r, Rectangle s)
int      rectclip(Rectangle *rp, Rectangle b)
void     combinerect(Rectangle *rp, Rectangle b)
int      Dx(Rectangle r)
int      Dy(Rectangle r)
Point    Pt(int x, int y)
Rectangle Rect(int x0, int y0, int x1, int y1)
Rectangle Rpt(Point p, Point q)

DESCRIPTION
The functions Pt, Rect and Rpt construct geometrical data types from their components.
Addpt returns the Point sum of its arguments: Pt(p.x+q.x, p.y+q.y). Subpt returns the
Point difference of its arguments: Pt(p.x–q.x, p.y–q.y). Mulpt returns the Point
Pt(p.x*a, p.y*a). Divpt returns the Point Pt(p.x/a, p.y/a).
Rectaddpt returns the Rectangle Rect(add(r.min, p), add(r.max, p)); rectspert returns
the Rectangle Rpt(sub(r.min, p), sub(r.max, p)).
Insetrect returns the Rectangle Rect(r.min.x+n, r.min.y+n, r.max.x–n, r.max.y–n).
Canonrect returns a rectangle with the same extent as r, canonicalized so that min.x ≤ max.x,
and min.y ≤ max.y.
Eqpt compares its argument Points and returns 0 if unequal, 1 if equal. Eqrect does the same for
its argument Rectangles.
Ptinrect returns 1 if p is a point within r, and 0 otherwise.
Rectinrect returns 1 if all the pixels in r are also in s, and 0 otherwise.
RectXrect returns 1 if r and s share any point, and 0 otherwise.
Rectclip clips in place the Rectangle pointed to by rp so that it is completely contained within b.
The return value is 1 if any part of *rp is within b. Otherwise, the return value is 0 and *rp is
unchanged.
Combinerect overwrites *rp with the smallest rectangle sufficient to cover all the pixels of *rp
and b.
The functions Dx and Dy give the width (Δx) and height (Δy) of a Rectangle. They are implemented
as macros.

SOURCE
/sys/src/libdraw

SEE ALSO
  graphics(2)
NAME
setupAESstate, aesCBCencrypt, aesCBCdecrypt, aesCTRencrypt, aesCTRdecrypt, setupAESXCBCstate, aesXCBCmac – advanced encryption standard (rijndael)

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <mp.h>
#include <libsec.h>

void aes_encrypt(ulong rk[], int Nr, uchar pt[16], uchar ct[16]);
void aes_decrypt(ulong rk[], int Nr, uchar ct[16], uchar pt[16]);
void setupAESstate(AESstate *s, uchar key[], int keybytes, uchar *ivec)
void aesCBCencrypt(uchar *p, int len, AESstate *s)
void aesCBCdecrypt(uchar *p, int len, AESstate *s)
void aesCTRencrypt(uchar *p, int len, AESstate *s)
void aesCTRdecrypt(uchar *p, int len, AESstate *s)
void setupAESXCBCstate(AESstate *s)
void aesX CBCmac(uchar *p, int len, AESstate *s)

DESCRIPTION
AES (a.k.a. Rijndael) has replaced DES as the preferred block cipher. Aes_encrypt and aes_decrypt are the block ciphers, corresponding to des(2)'s block_cipher. SetupAESstate, aesCBCencrypt, and aesCBCdecrypt implement cipher–block–chaining encryption. AesCTRencrypt and aesCTRdecrypt implement counter mode, per RFC 3686; they are identical operations. setupAESXCBCstate and aesX CBCmac implement AES XCBC message authentication, per RFC 3566. All ciphering is performed in place. Key bytes should be 16, 24, or 32. The initialization vector ivec of AESbsize bytes should be random enough to be unlikely to be reused but does not need to be cryptographically strongly unpredictable.

SOURCE
/sys/src/libsec

SEE ALSO
aescbc in sectore(1), mp(2), blowfish(2), des(2),dsa(2), elgamal(2), rc4(2), rsa(2), sechash(2), prime(2), rand(2)

BUGS
The functions aes_encrypt, aes_decrypt, aesCTRencrypt, aesCTRdecrypt, setupAESXCBCstate, and aesX CBCmac have not yet been verified by running test vectors through them.
NAME
allocimage, allocimagemix, freeimage, nameimage, namedimage, setalpha, loadimage, cloadimage, unloadimage, readimage, writeimage, bytesperline, wordsperline - allocating, freeing, reading, writing images

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>

Image *allocimage(Display *d, Rectangle r,
ulong chan, int repl, int col)

Image *allocimagemix(Display *d, ulong one, ulong three)

void freeimage(Image *i)

int nameimage(Image *i, char *name, int in)

Image *namedimage(Display *d, char *name)

ulong setalpha(ulong color, uchar alpha)

int loadimage(Image *i, Rectangle r, uchar *data, int ndata)

int cloadimage(Image *i, Rectangle r, uchar *data, int ndata)

int unloadimage(Image *i, Rectangle r, uchar *data, int ndata)

Image *readimage(Display *d, int fd, int dolock)

int writeimage(int fd, Image *i, int dolock)

int bytesperline(Rectangle r, int d)

int wordsperline(Rectangle r, int d)

enum
{
    DOpaque = 0xFFFFFFFF,
    DTransparent = 0x00000000,
    DBlack = 0x000000FF,
    DWhite = 0xFFFFFFFF,
    DRed = 0xFF0000FF,
    DGreen = 0x00FF00FF,
    DBlue = 0x0000FFFF,
    DCyan = 0x00FFFFFF,
    DMagenta = 0xFF00FFFF,
    DYellow = 0xFFFF00FF,
    DPaleyellow = 0xFFFFAAFF,
    DDarkyellow = 0xEEEE9EFF,
    DDarkgreen = 0x448844FF,
    DPalegreen = 0xAAFFAAFF,
    DMedgreen = 0x88CC88FF,
    DDarkblue = 0x000055FF,
    DPalebluegreen = 0x008888FF,
    DGreyblue = 0x005DBBFF,
    DPalegreyblue = 0x4993DDFF,
    DPurpleblue = 0x8888CCFF,
    DNotacolor = 0xFFFFFF00,
}

289
DNofill = DNotacolor,
};

DESCRIPTION
A new Image on Display d is allocated with allocimage; it will have the rectangle, pixel channel format, and replication flag given by its arguments. Convenient pixel channels like GREY1, GREY2, CMAP8, RGB16, RGB24, and RGBA32 are predefined. All the new image’s pixels will have initial value col. If col is DNofill, no initialization is done. Representative useful values of color are predefined: DB1ack, DWhite, DRed, and so on. Colors are specified by 32-bit numbers comprising, from most to least significant byte, 8-bit values for red, green, blue, and alpha. The values correspond to illumination, so 0 is black and 255 is white. Similarly, for alpha 0 is transparent and 255 is opaque. The id field will have been set to the identifying number used by /dev/draw (see draw(3)), and the cache field will be zero. If repl is true, the clip rectangle is set to a very large region; if false, it is set to r. The depth field will be set to the number of bits per pixel specified by the channel descriptor (see image(6)). Allocimage returns 0 if the server has run out of image memory.

Allocimagemix is used to allocate background colors. On 8-bit color-mapped displays, it returns a 2×2 replicated image with one pixel colored the color one and the other three with three. (This simulates a wider range of tones than can be represented by a single pixel value on a color-mapped display.) On true color displays, it returns a 1×1 replicated image whose pixel is the result of mixing the two colors in a one to three ratio.
Freimage frees the resources used by its argument image.

Nameimage publishes in the server the image i under the given name. If in is non–zero, the image is published; otherwise i must be already named name and it is withdrawn from publication. Namedimage returns a reference to the image published under the given name on Display d. These routines permit unrelated applications sharing a display to share an image; for example they provide the mechanism behind getwindow (see graphics(2)).

The RGB values in a color are premultiplied by the alpha value; for example, a 50% red is 0x7F00007F not 0xFF00007F. The function setalpha performs the alpha computation on a given color, ignoring its initial alpha value, multiplying the components by the supplied alpha. For example, to make a 50% red color value, one could execute setalpha(DRed, 0x7F).

The remaining functions deal with moving groups of pixel values between image and user space or external files. There is a fixed format for the exchange and storage of image data (see image(6)).

Unloadimage reads a rectangle of pixels from image i into data, whose length is specified by ndata. It is an error if ndata is too small to accommodate the pixels.
Loadimage replaces the specified rectangle in image i with the ndata bytes of data.
The pixels are presented one horizontal line at a time, starting with the top–left pixel of r. In the data processed by these routines, each scan line starts with a new byte in the array, leaving the last byte of the previous line partially empty, if necessary. Pixels are packed as tightly as possible within data, regardless of the rectangle being extracted. Bytes are filled from most to least significant bit order, as the x coordinate increases, aligned so x=0 would appear as the leftmost pixel of its byte. Thus, for depth 1, the pixel at x offset 165 within the rectangle will be in a data byte at bit–position 0x04 regardless of the overall rectangle: 165 mod 8 equals 5, and 0x80 >> 5 equals 0x04.

Cloadimage does the same as loadimage, but for ndata bytes of compressed image data (see image(6)). On each call to cloadimage, the data must be at the beginning of a compressed data block, in particular, it should start with the y coordinate and data length for the block.
Loadimage, cloadimage, and unloadimage return the number of bytes copied.

Readimage creates an image from data contained in an external file (see image(6) for the file format); fd is a file descriptor obtained by opening such a file for reading. The returned image is allocated using allocimage. The dolock flag specifies whether the Display should be synchronized for multithreaded access; single–threaded programs can leave it zero.
Writeimage writes image i onto file descriptor fd, which should be open for writing. The format is as described for readimage.
Readimage and writeimage do not close fd.

Bytesperline and wordsperline return the number of bytes or words occupied in memory by one scan line of rectangle r in an image with d bits per pixel.

EXAMPLE
To allocate a single-pixel replicated image that may be used to paint a region red,
   red = allocimage(display, Rect(0, 0, 1, 1), RGB24, 1, DRed);

SOURCE
/sys/src/libdraw

SEE ALSO
graphics(2), draw(2), draw(3), image(6)

DIAGNOSTICS
These functions return pointer 0 or integer -1 on failure, usually due to insufficient memory.
   May set errstr.

BUGS
Depth must be a divisor or multiple of 8.
NAME
ARGBEGIN, ARGEND, ARGC, ARGF, EARGF – process option letters from argv

SYNOPSIS
#include <u.h>
#include <libc.h>
ARGBEGIN {
  char *ARGF();
  char *EARGF(code);
  Rune ARGC();
} ARGEND
extern char *argv0;

DESCRIPTION
These macros assume the names argc and argv are in scope; see exec(2). ARGBEGIN and ARGEND
surround code for processing program options. The code should be the cases of a C switch on
option characters; it is executed once for each option character. Options end after an argument
--, before an argument --, or before an argument that doesn’t begin with --.

The function macro ARGC returns the current option character, as an integer.

The function macro ARGF returns the current option argument: a pointer to the rest of the option
string if not empty, or the next argument in argv if any, or 0. ARGF must be called just once for
each option argument. The macro EARGF is like ARGF but instead of returning zero runs code and,
if that returns, calls abort(2). A typical value for code is usage(), as in EARGF(usage()).

After ARGBEGIN, argv0 is a copy of argv[0] (conventionally the name of the program).
After ARGEND, argv points at a zero–terminated list of the remaining argc arguments.

EXAMPLE
This C program can take option b and option f, which requires an argument.
#include <u.h>
#include <libc.h>
void
main(int argc, char *argv[])
{
  char *f;
  print("%s", argv[0]);
  ARGBEGIN {
    case 'b':
      print(" -b");
      break;
    case 'f':
      print(" -f(%s)", (f=ARGF())? f: "no arg");
      break;
    default:
      print(" badflag('%c')", ARGC());
  } ARGEND
  print(" %d args:", argc);
  while(*argv)
    print(" '%s'", *argv++);
  exits(nil);
}

Here is the output from running the command prog -b file1 -r -f file2 arg1 arg2
prog -b -f(file1) badflag('r') -f(file2) 2 args: 'arg1' 'arg2'

SOURCE
/sys/include/libc.h
SEE ALSO

getflags(8)
NAME
add3, sub3, neg3, div3, mul3, eqpt3, closept3, dot3, cross3, len3, dist3, unit3, midpt3, lerp3, reflect3, nearseg3, pldist3, vdiv3, vrem3, pn2f3, ppp2f3, fff2p3, pdiv4, add4, sub4 – operations on 3-d points and planes

SYNOPSIS
#include <draw.h>
#include <geometry.h>
Point3 add3(Point3 a, Point3 b)
Point3 sub3(Point3 a, Point3 b)
Point3 neg3(Point3 a)
Point3 div3(Point3 a, double b)
Point3 mul3(Point3 a, double b)
int eqpt3(Point3 p, Point3 q)
int closept3(Point3 p, Point3 q, double eps)
double dot3(Point3 p, Point3 q)
Point3 cross3(Point3 p, Point3 q)
double len3(Point3 p)
double dist3(Point3 p, Point3 q)
Point3 unit3(Point3 p)
Point3 midpt3(Point3 p, Point3 q)
Point3 lerp3(Point3 p, Point3 q, double alpha)
Point3 reflect3(Point3 p, Point3 p0, Point3 p1)
Point3 nearseg3(Point3 p0, Point3 p1, Point3 testp)
double pldist3(Point3 p, Point3 p0, Point3 p1)
double vdiv3(Point3 a, Point3 b)
Point3 vrem3(Point3 a, Point3 b)
Point3 pn2f3(Point3 p, Point3 n)
Point3 ppp2f3(Point3 p0, Point3 p1, Point3 p2)
Point3 fff2p3(Point3 f0, Point3 f1, Point3 f2)
Point3 pdiv4(Point3 a)
Point3 add4(Point3 a, Point3 b)
Point3 sub4(Point3 a, Point3 b)

DESCRIPTION
These routines do arithmetic on points and planes in affine or projective 3–space. Type Point3 is

typedef struct Point3 Point3;
struct Point3{
    double x, y, z, w;
};

Routines whose names end in 3 operate on vectors or ordinary points in affine 3–space, represented by their Euclidean \((x, y, z)\) coordinates. (They assume \(w=1\) in their arguments, and set \(w=1\) in their results.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add3</td>
<td>Add the coordinates of two points.</td>
</tr>
<tr>
<td>sub3</td>
<td>Subtract coordinates of two points.</td>
</tr>
</tbody>
</table>
neg3 Negate the coordinates of a point.
mul3 Multiply coordinates by a scalar.
div3 Divide coordinates by a scalar.
eqpt3 Test two points for exact equality.
closept3 Is the distance between two points smaller than eps?
dot3 Dot product.
cross3 Cross product.
len3 Distance to the origin.
dist3 Distance between two points.
unit3 A unit vector parallel to p.
midpt3 The midpoint of line segment pq.
lerp3 Linear interpolation between p and q.
reflect3 The reflection of point p in the segment joining p0 and p1.
nearseg3 The closest point to testp on segment p0 p1.
pldist3 The distance from p to segment p0 p1.
vdiv3 Vector divide — the length of the component of a parallel to b, in units of the length of b.
vrem3 Vector remainder — the component of a perpendicular to b. Ignoring roundoff, we have eqpt3(add3(mul3(b, vdiv3(a, b)), vrem3(a, b)), a).

The following routines convert amongst various representations of points and planes. Planes are represented identically to points, by duality, a point p is on a plane q whenever \( p.x \times q.x + p.y \times q.y + p.z \times q.z + p.w \times q.w = 0 \). Although when dealing with affine points we assume \( p.w = 1 \), we can’t make the same assumption for planes. The names of these routines are extra-cryptic. They contain an ‘f’ (for ‘face’) to indicate a plane, p for a point and n for a normal vector. The number 2 abbreviates the word ‘to.’ The number 3 reminds us, as before, that we’re dealing with affine points. Thus pn2f3 takes a point and a normal vector and returns the corresponding plane.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pn2f3</td>
<td>Compute the plane passing through p with normal n.</td>
</tr>
<tr>
<td>ppp2f3</td>
<td>Compute the plane passing through three points.</td>
</tr>
<tr>
<td>fff2p3</td>
<td>Compute the intersection point of three planes.</td>
</tr>
</tbody>
</table>

The names of the following routines end in 4 because they operate on points in projective 4–space, represented by their homogeneous coordinates.

pdiv4 Perspective division. Divide p.w into p’s coordinates, converting to affine coordinates. If p.w is zero, the result is the same as the argument.

add4 Add the coordinates of two points.
sub4 Subtract the coordinates of two points.

SOURCE
/sys/src/libgeometry

SEE ALSO
matrix(2)
NAME
  assert - check program invariants

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  #define assert(cond) if(cond);else _assert("cond")
  void _assert(char* cond)

DESCRIPTION
  Assert is a preprocessor macro that (via _assert) prints a message and calls abort when cond is false.

SOURCE
  /sys/src/libc/port/_assert.c
NAME
atof, atoi, atol, atoll, charstod, strtod, strtol, strtoll, strtoul, strtoull – convert text to numbers

SYNOPSIS
#include <u.h>
#include <libc.h>
double atof(char *nptr)
int atoi(char *nptr)
long atol(char *nptr)
vlong atoll(char *nptr)
double charstod(int (*f)(void *), void *a)
double strtod(char *nptr, char **rptr)
long strtol(char *nptr, char **rptr, int base)
vlong strtoll(char *nptr, char **rptr, int base)
ulong strtoul(char *nptr, char **rptr, int base)
uvlong strtoull(char *nptr, char **rptr, int base)

DESCRIPTION
Atof, atoi, atol, and atoll convert a string pointed to by nptr to floating, integer, long integer, and long long integer (vlong) representation respectively. The first unrecognized character ends the string. Leading C escapes are understood, as in strtol with base zero (described below).

Atof recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optionally signed integer.

Atoi and atol recognize an optional string of tabs and spaces, then an optional sign, then a string of decimal digits.

Strtod, strtol, strtof, strtoul, and strtoull behave similarly to atof and atol and, if rptr is not zero, set *rptr to point to the input character immediately after the string converted.

Strtol, strtoll, strtoul, and strtoull interpret the digit string in the specified base, from 2 to 36, each digit being less than the base. Digits with value over 9 are represented by letters, a–z or A–Z. If base is 0, the input is interpreted as an integral constant in the style of C (with no suffixed type indicators): numbers are octal if they begin with 0, hexadecimal if they begin with 0x or 0X, otherwise decimal.

Charstod interprets floating point numbers in the manner of atof, but gets successive characters by calling (*f)(a). The last call to f terminates the scan, so it must have returned a character that is not a legal continuation of a number. Therefore, it may be necessary to back up the input stream one character after calling charstod.

SOURCE
/sys/src/libc/port

SEE ALSO
fscanf(2)

DIAGNOSTICS
Zero is returned if the beginning of the input string is not interpretable as a number; even in this case, rptr will be updated.

BUGS
Atoi, atol, and atoll accept octal and hexadecimal numbers in the style of C, contrary to the ANSI specification.
NAME
aic, adec, cas, casv, casp, loadlink, storecond, _tas – atomic RMW operations

SYNOPSIS
#include <u.h>
#include <libc.h>
long ainc(long *addr);
long adec(long *addr);
int cas(int *addr, int ov, int nv);
int casv(u64int *addr, u64int ov, u64int nv);
int casp(void **addr, void *ov, void *nv);
int _tas(ulong *addr);
ulong loadlink(ulong*);
int storecond(ulong*, ulong);

DESCRIPTION
Ainc atomically increments the value pointed to by addr and returns the new value.
Adec atomically decrements the value pointed to by addr and returns the new value.
Cas, casv and casp implement Compare–and–Swap on, respectively, int, vlong and void* values. The availability of these functions depends on the CPU architecture: Pentium III and later, as well as AMD64 have 64–bit CAS instructions. Other architectures don’t. ARM–5 processors and earlier do not have CAS (nor have they Load–Linked or Store–Conditional). These instructions are, however, emulated by the Plan 9 kernel. All other architectures have 32–bit CAS.
_tas implements Test–and–Set, which is available on all architectures and used for the implementation of kernel locks (see lock(2) and thread(2)).
Loadlink and storecond access the load–linked and store–conditional instructions present on MIPS (LL/SC), ARM (Strex/Ldrex), PowerPC (LWAR/STWCC), Alpha (MOVL, MOVLC). These are not present on Pentium or AMD64. On the architectures that have load–linked and store–conditional, these are used to implement compare–and–swap.

SOURCE
/sys/src/libc/*/atom.s
/sys/src/libc/*/tas.s

SEE ALSO
lock(2), semacquire(2), thread(2)

DIAGNOSTICS
The CAS functions, _tas, and storecond return 0 for failure and 1 for success.
NAME
amount, newns, addns, login, noworld, auth_proxy, fauth_proxy, auth_allocrpc, auth_freerp, auth_rpc, auth_getkey, amount_getkey, auth_freeAI, auth_chuid, auth_challenge, auth_response, auth_freechal, auth_respond, auth_userpasswd, auth_getuserpasswd, auth_getinfo – routines for authenticating users

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <auth.h>

int newns(char *user, char *nsfile);
int addns(char *user, char *nsfile);
int amount(int fd, char *old, int flag, char *aname);
int login(char *user, char *password, char *namespace);
int noworld(char *user);
AuthInfo* auth_proxy(int fd, AuthGetkey *getkey, char *fmt, ...);
AuthInfo* fauth_proxy(int fd, AuthRpc *rpc, AuthGetkey *getkey, char *params);
AuthRcp* auth_allocrpc(int afd);
void auth_freerp(AuthRcp *rpc);
uint auth_rpc(AuthRcp *rpc, char *verb, void *a, int n);
int auth_getkey(char *proto, char *dom);
int (*amount_getkey)(char*, char*);
void auth_freeAI(AuthInfo *ai);
int auth_chuid(AuthInfo *ai, char *ns);
Chalstate* auth_challenge(char *fmt, ...);
AuthInfo* auth_response(Chalstate*);
void auth_freechal(Chalstate*);
int auth_respond(void *chal, uint nchal, char *user, uint nuser, void *resp, uint nresp, AuthGetkey *getkey, char *fmt, ...);
AuthInfo* auth_userpasswd(char*user, char*password);
UserPasswd* auth_getuserpasswd(AuthGetkey *getkey, char*fmt, ...);
AuthInfo* auth_getinfo(int fd);

DESCRIPTION
This library, in concert with factotum(4), is used to authenticate users. It provides the primary interface to factotum.

Newns builds a name space for user. It opens the file nsfile (/lib/namespace is used if nsfile is null), copies the old environment, erases the current name space, sets the environment variables user and home, and interprets the commands in nsfile. The format of nsfile is described in namespace(6).

Addns also interprets and executes the commands in nsfile. Unlike newns it applies the command to the current name space rather than starting from scratch.

Amount is like mount but performs any authentication required. It should be used instead of mount whenever the file server being mounted requires authentication. See bind(2) for a definition of the arguments to mount and amount.

Login changes the user id of the process user and recreates the namespace using the file namespace (default /lib/namespace). It uses auth_userpasswd and auth_chuid.
Noworld returns 1 if the user is in the group noworld in /adm/users. Otherwise, it returns 0. Noworld is used by telnetd and ftpd to provide sandboxed access for some users.

The following routines use the AuthInfo structure returned after a successful authentication by factotum(4).

```c
typedef struct
{
    char *cuid; /* caller id */
    char *suid; /* server id */
    char *cap; /* capability */
    int nsecret; /* length of secret */
    uchar *secret; /* secret */
} AuthInfo;
```

The fields cuid and suid point to the authenticated ids of the client and server. Cap is a capability returned only to the server. It can be passed to the cap(3) device to change the user id of the process. Secret is an nsecret-byte shared secret that can be used by the client and server to create encryption and hashing keys for the rest of the conversation.

Auth_proxy proxies an authentication conversation between a remote server reading and writing fd and a factotum file. The factotum file used is /mnt/factotum/rpc. An sprint (see print(2)) of fmt and the variable arg list yields a key template (see factotum(4)) specifying the key to use. The template must specify at least the protocol (proto=xxx) and the role (either role=client or role=server). Auth_proxy either returns an allocated AuthInfo structure, or sets the error string and returns nil.

Fauth_proxy can be used instead of auth_proxy if a single connection to factotum will be used for multiple authentications. This is necessary, for example, for newns which must open the factotum file before wiping out the namespace. Fauth_proxy takes as an argument a pointer to an AuthRPC structure which contains an fd for an open connection to factotum in addition to storage and state information for the protocol. An AuthRPC structure is obtained by calling auth_allocrpc with the fd of an open factotum connection. It is freed using auth_freerpc. Individual commands can be sent to factotum(4) by invoking auth_rpc.

Both auth_proxy and fauth_proxy take a pointer to a routine, getkey, to invoke should factotum not possess a key for the authentication. If getkey is nil, the authentication fails. Getkey is called with a key template for the desired key. We have provided a generic routine, auth_getkey, which queries the user for the key information and passes it to factotum. This is the default for the global variable, amount_getkey, which holds a pointer to the key prompting routine used by amount.

Auth_chuid uses the cuid and cap fields of an AuthInfo structure to change the user id of the current process and uses ns, default /lib/namespace, to build it a new name space.

Auth_challenge and auth_response perform challenge/response protocols with factotum. State between the challenge and response phase are kept in the Chalstate structure:

```c
struct Chalstate
{
    char *user;
    char chal[MAXCHLEN];
    int nchal;
    void *resp;
    int nresp;

    /* for implementation only */
    int afd;
    AuthRcp *rpc;
    char userbuf[MAXNAMELEN];
    int userinchal;
};
```

Auth_challenge requires a key template generated by an sprint of fmt and the variable arguments. It must contain the protocol (proto=xxx) and depending on the protocol, the user name (user=xxx). P9cr and vnc expect the user specified as an attribute in the key template and
apop, cram, and chap expect it in the user field of the arg to auth_response. For all protocols, the response is returned to auth_response in the resp field of the Chalstate. Chalstate.nresp must be the length of the response.

Supply to auth_respond a challenge string and the fmt and args specifying a key, and it will use factotum to return the proper user and response.

Auth_userpasswd verifies a simple user/password pair. Auth_getuserpasswd retrieves a user/password pair from factotum if permitted:

```c
typedef struct UserPasswd {
    char *user;
    char *passwd;
} UserPasswd;
```

Auth_getinfo reads an AuthInfo message from fd and converts it into a structure. It is only used by the other routines in this library when communicating with factotum.

Auth_freeAl is used to free an AuthInfo structure returned by one of these routines. Similary auth_freechal frees a challenge/response state.

**SOURCE**
/sys/src/libauth

**SEE ALSO**
factotum(4), authsrv(2), bind(2)

**DIAGNOSTICS**
These routines set errstr.
NAME
authdial, passtokey, nvcs, readnram, convT2M, convM2T, convTR2M, convM2TR, convA2M, convM2A, convPR2M, convM2PR, _asgetticket, _asrdresp – routines for communicating with authentication servers

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <authsrv.h>

int authdial(char *netroot, char *ad);
int passtokey(char key[DESKEYLEN], char *password)
uchar nvcs(void *mem, int len)
int readnram(Nvrsafe *nv, int flag);
int convT2M(Ticket *t, char *msg, char *key)
void convM2T(char *msg, Ticket *t, char *key)
int convA2M(Authenticator *a, char *msg, char *key)
void convM2A(char *msg, Authenticator *a, char *key)
int convTR2M(Ticketreq *tr, char *msg)
void convM2TR(char *msg, Ticketreq *tr)
int convPR2M(Passwordreq *pr, char *msg, char *key)
void convM2PR(char *msg, Passwordreq *pr, char *key)
int _asgetticket(int fd, char *trbuf, char *tbuf);
int _asrdresp(int fd, char *buf, int len);

DESCRIPTION
Authdial dials an authentication server over the network rooted at net, default /net. The authentication domain, ad, specifies which server to call. If ad is non-nil, the connection server cs (see ndb(8)) is queried for an entry which contains authdom=ad or dom=ad, the former having precedence, and which also contains an auth attribute. If it finds neither, it tries p9auth. ad in DNS as the authentication server. The string dialed is then netroot!server!ticket where server is the value of the auth attribute. If no entry is found, the error string is set to "no authentication server found" and -1 is returned. If authdom is nil, the string netroot!auth!ticket is used to make the call.

Passtokey converts password into a DES key and stores the result in key. It returns 0 if password could not be converted, and 1 otherwise.

Readnram reads authentication information into the structure:

struct Nvrsafe
{
    char    machkey[DESKEYLEN]; /* was file server's authid's des key */
    uchar   machsum;
    char    authkey[DESKEYLEN]; /* authid's des key from password */
    uchar   authsum;
    /*
    * file server config string of device holding full configuration;
    * secstore key on non-file-servers.
    */
    char    config[CONFIGLEN];
    uchar   configsum;
    char    authid[ANAMELEN]; /* auth userid, e.g., bootes */
    uchar   authidsum;
    char    authdom[DOMLEN]; /* auth domain, e.g., cs.bell-labs.com */
    uchar   authdomsum;
}
On Sparc, MIPS, and SGI machines this information is in non-volatile ram, accessible in the file 
#r/nvram. On x86s and Alphas readnvram successively opens the following areas stopping with 
the first to succeed:
- the partition named by the $nvram environment variable (commonly set via plan9.ini(8))
- the partition #S/sdC0/nvram
- a file called plan9.nvr in the partition #S/sdC0/9fat
- the partition #S/sd00/nvram
- a file called plan9.nvr in the partition #S/sd00/9fat
- a file called plan9.nvr on a DOS floppy in drive 0
- a file called plan9.nvr on a DOS floppy in drive 1
The nvcsums of the fields machkey, authid, and authdom must match their respective check-
sum or that field is zeroed. If flag is NVwrite or at least one checksum fails and flag is 
NVwriteonerr, readnvram will prompt for new values on #c/cons and then write them back 
to the storage area. If flag is NVwritemem, readnvram will write the values in *nv back to the 
storage area.
ConvT2M, convA2M, convTR2M, and convPR2M convert tickets, authenticators, ticket requests, and 
password change request structures into transmittable messages. ConvM2T, convM2A, convM2TR, 
and convM2PR are used to convert them back. Key is used for encrypting the message before 
transmission and decrypting after reception.
The routine _asgetresp receives either a character array or an error string. On error, it sets errstr 
and returns −1. If successful, it returns the number of bytes received.
The routine _asgetticket sends a ticket request message and then uses _asgetresp to receive an 
answer.

SOURCE
/sys/src/libauthsrv

SEE ALSO
passwd(1), cons(3), dial(2), authsrv(6),

DIAGNOSTICS
These routines set errstr. Integer-valued functions return −1 on error.
NAME
mkavltree, insertavl, lookupavl, deleteavl, avlwalk, avlnext, avlprev, endwalk – AVL tree routines

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <avl.h>
typedef struct Avl Avl;
struct Avl
{
    Avl  *p;        /* parent */
    Avl  *n[2];     /* children */
    int   bal;      /* balance bits */
};
Avl *avlnext(Avlwalk *walk);
Avl *avlprev(Avlwalk *tree);
void deleteavl(Avltree *tree, Avl *key, Avl **oldp);
void endwalk(Avlwalk *walk);
void insertavl(Avltree *tree, Avl *new, Avl **oldp);
Avl *lookupavl(Avltree *tree, Avl *key);
Avl *searchavl(Avltree *tree, Avl *key, int neighbor);
Avltree *mkavltree(int (*cmp)(Avl*, Avl*));

DESCRIPTION
An AVL tree is a self–balancing binary search tree. These routines allow creation and maintenance of in–memory AVL trees.

An empty AVL tree is created by calling mkavltree with a comparison function as argument. This function should take two pointers to Avl objects and return −1, 0 or 1 as the first is respectively less than, equal to, or greater than, the second. Insertavl adds a new tree node into tree. If oldp is non-nil upon return, it points to storage for an old node with the same key that may now be freed. Lookupavl returns the tree node that matches key by tree’s comparison function, or nil if none. Searchavl returns the tree node that matches key by tree’s comparison function, if it exists. If it does not, and neighbor is positive, it returns the nearest node whose key is greater or nil if there is none and, if neighbor is negative, it returns the nearest node whose key is less or nil if there is none. It is an error to set neighbor to values other than −1, 0, or +1.

Deleteavl removes the node matching key from tree; oldp is handled as per insertavl.

Avlwalk returns a pointer to a newly–allocated Avlwalk object. Endwalk frees such an object. Avlnext and avlprev walk the tree associated with walk, returning the next (respectively, previous) tree node in the comparison order defined by the comparison function associated with the tree associated with walk.

EXAMPLES
Intended usage seems to be to make an anonymous Avl the first member of the application’s tree–node structure, then pass these routines tree–node pointers instead of Avl*’s.

typedef struct Node {
    Avl;
    uchar  score[VtScoreSize];
    int    type;
} Node;
Avltree *tree;
Avl *res;
Node *np;
...  
    res = lookupavl(tree, np);

SOURCE
/sys/src/libavl
SEE ALSO

DIAGNOSTICS
Functions returning pointers return nil on error.
NAME
binalloc, bingroup, binfree – grouped memory allocation

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <bin.h>

typedef struct BinBin;
    void *binalloc(Bin **bp, ulong size, int clr);
    void *bingrow(Bin **bp, void *op, ulong osize,
                   ulong size, int clr);
    void binfree(Bin **bp);

DESCRIPTION
These routines provide simple grouped memory allocation and deallocation. Items allocated with
binalloc are added to the Bin pointed to by bp. All items in a bin may be freed with one call to
binfree; there is no way to free a single item.

Binalloc returns a pointer to a new block of at least size bytes. The block is suitably aligned for
storage of any type of object. No two active pointers from binalloc will have the same value. The
call binalloc(0) returns a valid pointer rather than null. If clr is non–zero, the allocated mem-
ory is set to 0; otherwise, the contents are undefined.

Bingroup is used to extend the size of a block of memory returned by binalloc. Bp must point to the
same bin group used to allocate the original block, and osize must be the last size used to allocate
or grow the block. A pointer to a block of at least size bytes is returned, with the same contents in
the first osize locations. If clr is non–zero, the remaining bytes are set to 0, and are undefined
otherwise. If op is nil, it and osize are ignored, and the result is the same as calling binalloc.

Binalloc and bingroup allocate large chunks of memory using malloc(2) and return pieces of these
chunks. The chunks are free’d upon a call to binfree.

SOURCE
/sys/src/libbin

SEE ALSO
malloc(2)

DIAGNOSTICS
binalloc and bingroup return 0 if there is no available memory.
NAME
bind, mount, unmount - change name space

SYNOPSIS
#include <u.h>
#include <libc.h>
int bind(char *name, char *old, int flag)
int mount(int fd, int afd, char *old, int flag, char *aname)
int unmount(char *name, char *old)

DESCRIPTION
Bind and mount modify the file name space of the current process and other processes in its name
space group (see fork(2)). For both calls, old is the name of an existing file or directory in the cur-
rent name space where the modification is to be made. The name old is evaluated as described in
intro(2), except that no translation of the final path element is done.

For bind, name is the name of another (or possibly the same) existing file or directory in the cur-
rent name space. After a successful bind call, the file name old is an alias for the object originally
named by name; if the modification doesn’t hide it, name will also still refer to its original file.
The evaluation of new happens at the time of the bind, not when the binding is later used.

The fd argument to mount is a file descriptor of an open network connection or pipe to a file
server, while afd is a authentication file descriptor as created by fauth(2) and subsequently authen-
ticated. If authentication is not required, afd should be −1. The old file must be a directory. After
a successful mount the file tree served (see below) by fd will be visible with its root directory hav-
ing name old.

The flag controls details of the modification made to the name space. In the following, new refers
to the file as defined by name or the root directory served by fd. Either both old and new files must
be directories, or both must not be directories. Flag can be one of:

MREPL Replace the old file by the new one. Henceforth, an evaluation of old will be trans-
lated to the new file. If they are directories (for mount, this condition is true by defi-
nition), old becomes a union directory consisting of one directory (the new file).

MBEFORE Both the old and new files must be directories. Add the constituent files of the new
directory to the union directory at old so its contents appear first in the union. After
an MBEFORE bind or mount, the new directory will be searched first when evaluating
file names in the union directory.

MAFTER Like MBEFORE but the new directory goes at the end of the union.

The flags are defined in <libc.h>. In addition, there is an MCREATE flag that can be OR’d with
any of the above. When a create system call (see open(2)) attempts to create in a union directory,
and the file does not exist, the elements of the union are searched in order until one is found with
MCREATE set. The file is created in that directory; if that attempt fails, the create fails.

Finally, the MCACHE flag, valid for mount only, turns on caching for files made available by the
mount. By default, file contents are always retrieved from the server. With caching enabled, the
kernel may instead use a local cache to satisfy read(5) requests for files accessible through this
mount point. The currency of cached data for a file is verified at each open(5) of the file from this
client machine.

With mount, the file descriptor fd must be open for reading and writing and prepared to respond
to 9P messages (see Section 5). After the mount, the file tree starting at old is served by a kernel
mnt(3) device. That device will turn operations in the tree into messages on fd. Aname selects
among different file trees on the server; the null string chooses the default tree.

The file descriptor fd is automatically closed by a successful mount call.

The effects of bind and mount can be undone by unmount. If name is zero, everything bound to or
mounted upon old is unbound or unmounted. If name is not zero, it is evaluated as described
above for bind, and the effect of binding or mounting that particular result on old is undone.

SOURCE
/sys/src/libc/9syscall
SEE ALSO
bind(1), intro(2), fcall(2), auth(2) (particularly amount), intro(5), mnt(3), srv(3)

DIAGNOSTICS
The return value is a positive integer (a unique sequence number) for success, -1 for failure.
These routines set errstr.

BUGS
Mount will not return until it has successfully attached to the file server, so the process doing a
mount cannot be the one serving.
NAME
Bopen, Binit, Binits, Brdline, Brdstr, Bgetc, Bgetrune, Bgetd, Bungetc, Bungetrune, Bread, Bseek,
Boffset, Bfildes, Blinelen, Bputc, Bputrune, Bprint, Bvprint, Bwrite, Bflush, Bterm, Bbuffered – buff-
fered input/output

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <bio.h>

Biobuf* Bopen(char *file, int mode)
int Binit(Biobuf *bp, int fd, int mode)
int Binits(Biobufhdr *bp, int fd, int mode, uchar *buf, int size)
int Bterm(Biobufhdr *bp)
int Bprint(Biobufhdr *bp, char *format, ...)
int Bvprint(Biobufhdr *bp, char *format, va_list arglist);
void* Brdline(Biobufhdr *bp, int delim)
char* Brdstr(Biobufhdr *bp, int delim, int nulldelim)
int Blinelen(Biobufhdr *bp)
vlong Boffset(Biobufhdr *bp)
int Bfildes(Biobufhdr *bp)
int Bgetc(Biobufhdr *bp)
long Bgetd(Biobufhdr *bp, double *d)
int Bungetc(Biobufhdr *bp)
int Bungetrune(Biobufhdr *bp)
vlong Bseek(Biobufhdr *bp, vlong n, int type)
int Bputc(Biobufhdr *bp, int c)
int Bputrune(Biobufhdr *bp, long c)
long Bread(Biobufhdr *bp, void *addr, long nbytes)
long Bwrite(Biobufhdr *bp, void *addr, long nbytes)
int Bflush(Biobufhdr *bp)
int Bbuffered(Biobufhdr *bp)

DESCRIPTION
These routines implement fast buffered I/O. I/O on different file descriptors is independent.
Bopen opens file for mode OREAD or creates for mode OWRITE. It calls malloc(2) to allocate a
buffer.

Binit initializes a standard size buffer, type Biobuf, with the open file descriptor passed in by the
user. Binits initializes a non-standard size buffer, type Biobufhdr, with the open file descriptor,
buffer area, and buffer size passed in by the user. Biobuf and Biobufhdr are related by the declara-
tion:

typedef struct Biobuf Biobuf;
struct Biobuf
{
    Biobufhdr;
    uchar b[Bungetsize+Bsize];
};
Arguments of types pointer to Biobuf and pointer to Biobufhdr can be used interchangeably in the following routines.

`Bopen`, `Binit`, or `Binit` should be called before any of the other routines on that buffer. `Bffildes` returns the integer file descriptor of the associated open file.

`Bterm` flushes the buffer for `bp` and returns `Bflush`'s return value. If the buffer was allocated by `Bopen`, the buffer is freed and the file is closed.

`Brdline` reads a string from the file associated with `bp` up to and including the first `delim` character. The delimiter at the end of the line is not altered, thus the returned string probably won’t be NUL–terminated. `Brdline` returns a pointer to the start of the line or 0 on end–of–file or read error. `Blinelen` returns the length (including the delimiter) of the most recent string returned by `Brdline`.

`Brdstr` returns a `malloc(2)`–allocated buffer containing the next line of input delimited by `delim`, terminated by a NUL (0) byte. Unlike `Brdline`, which returns when its buffer is full even if no delimiter has been found, `Brdstr` will return an arbitrarily long line in a single call. If `nulldelim` is set, the terminal delimiter will be overwritten with a NUL. After a successful call to `Brdstr`, the return value of `Blinelen` will be the length of the returned buffer, excluding the NUL.

`Bgetc` returns the next character from `bp`, or a negative value at end of file. `Bungetc` may be called immediately after `Bgetc` to allow the same character to be reread.

`Bgetrune` calls `Bgetc` to read the bytes of the next UTF sequence in the input stream and returns the value of the rune represented by the sequence. It returns a negative value at end of file. `Bungetrune` may be called immediately after `Bgetrune` to allow the same UTF sequence to be reread as either bytes or a rune. `Bungetrune` and `Bgetrune` may back up a maximum of five bytes.

`Bgetd` uses `charstod` (see `atof(2)`) and `Bgetc` to read the formatted floating–point number in the input stream, skipping initial blanks and tabs. The value is stored in `*d`.

`Bread` reads `nbytes` of data from `bp` into memory starting at `addr`. The number of bytes read is returned on success and a negative value is returned if a read error occurred.

`Bseek` applies `seek(2)` to `bp`. It returns the new file offset. `Boffset` returns the file offset of the next character to be processed.

`Bputc` outputs the low order 8 bits of `c` on `bp`. If this causes a `write` to occur and there is an error, a negative value is returned. Otherwise, a zero is returned.

`Bputrune` calls `Bputc` to output the low order 16 bits of `c` as a rune in UTF format on the output stream.

`Bprint` is a buffered interface to `print(2)`. If this causes a `write` to occur and there is an error, a negative value (`Beof`) is returned. Otherwise, `Bprint` returns the number of bytes written. `Bvprint` does the same except it takes as argument a `va_list` parameter, so it can be called within a variadic function.

`Bwrite` outputs `nbytes` of data starting at `addr` to `bp`. If this causes a `write` to occur and there is an error, a negative value is returned. Otherwise, the number of bytes written is returned.

`Bflush` causes any buffered output associated with `bp` to be written. The return is as for `Bputc`. `Bflush` is called on exit for every buffer still open for writing.

`Bbuffered` returns the number of bytes in the buffer. When reading, this is the number of bytes still available from the last read on the file; when writing, it is the number of bytes ready to be written.

**SOURCE**

`/sys/src/libbio`

**SEE ALSO**

`open(2)`, `print(2)`, `exits(2)`, `utf(6)`.

**DIAGNOSTICS**

`Bio` routines that return integers yield `Beof` if `bp` is not the descriptor of an open file. `Bopen` returns zero if the file cannot be opened in the given mode. All routines set `errstr` on error.

**BUGS**

`Brdline` returns an error on strings longer than the buffer associated with the file and also if the
end-of-file is encountered before a delimiter. *Blinelen* will tell how many characters are available in these cases. In the case of a true end-of-file, *Blinelen* will return zero. At the cost of allocating a buffer, *Brdstr* sidesteps these issues.

Only the low byte of *Brdstr*’s *delim* is examined, so *delim* cannot be an arbitrary rune.

The data returned by *Brdline* may be overwritten by calls to any other *bio* routine on the same *bp*. 
NAME
setupBFstate, bfCBCcencrypt, bfCBCdencrypt, bfECBencrypt, bfECBdencrypt – blowfish encryption

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <mp.h>
#include <libsec.h>

void setupBFstate(BFstate *s, uchar key[], int keybytes,
                  uchar *ivec)

void bfCBCcencrypt(uchar *data, int len, BFstate *s)

void bfCBCdencrypt(uchar *data, int len, BFstate *s)

void bfECBencrypt(uchar *data, int len, BFstate *s)

void bfECBdencrypt(uchar *data, int len, BFstate *s)

DESCRIPTION
Blowfish is Bruce Schneier’s symmetric block cipher. It supports variable length keys from 32 to
448 bits and has a block size of 64 bits. Both CBC and ECB modes are supported.

setupBFstate takes a BFstate structure, a key of at most 56 bytes, the length of the key in bytes,
and an initialization vector of 8 bytes (set to all zeroes if argument is nil). The encryption and
decryption functions take a BFstate structure, a data buffer, and a length, which must be a multiple
of eight bytes as padding is currently unsupported.

SOURCE
/sys/src/libsec

SEE ALSO
mp(2), aes(2), des(2), dsa(2), elgamal(2), rc4(2), rsa(2), sechash(2), prime(2), rand(2)
NAME
  brk, sbrk – change memory allocation

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  int brk(void *addr)
  void* sbrk(ulong incr)

DESCRIPTION
  Brk sets the system’s idea of the lowest bss location not used by the program (called the break) to 
  addr rounded up to the next multiple of 8 bytes. Locations not less than addr and below the stack 
  pointer may cause a memory violation if accessed.

  In the alternate function sbrk, incr more bytes are added to the program’s data space and a 
  pointer to the start of the new area is returned. Rounding occurs as with brk.

  When a program begins execution via exec the break is set at the highest location defined by the 
  program and data storage areas. Ordinarily, therefore, only programs with growing data areas 
  need to use brk. A call to sbrk with a zero argument returns the lowest address in the dynamic 
  segment.

SOURCE
  /sys/src/libc/9sys/sbrk.c

SEE ALSO
  intro(2), malloc(2), segattach(2), segbrk(2)

DIAGNOSTICS
  These functions set errstr.
  The error return from sbrk is (void*)−1.
NAME
  cachechars, agefont, loadchar, Subfont, Fontchar, Font – font utilities

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>

int cachechars(Font *f, char **s, Rune **r, ushort *c, int max,
   int *widp, char **sfname)

int loadchar(Font *f, Rune r, Cacheinfo *c, int h,
   int noclr, char **sfname)

void agefont(Font *f)

DESCRIPTION
A Font may contain too many characters to hold in memory simultaneously. The graphics library
and draw device (see draw(3)) cooperate to solve this problem by maintaining a cache of recently
used character images. The details of this cooperation need not be known by most programs:
initdraw and its associated font variable, openfont, stringwidth, string, and freefont are sufficient
for most purposes. The routines described below are used internally by the graphics library to
maintain the font cache.

A Subfont is a set of images for a contiguous range of characters, stored as a single image with
the characters placed side-by-side on a common baseline. It is described by the following data
structures.

typedef
  struct Fontchar {
  int  x;    /* left edge of bits */
  uchar top;  /* first non-zero scan-line */
  uchar bottom;  /* last non-zero scan-line */
  char left;  /* offset of baseline */
  uchar width;  /* width of baseline */
  } Fontchar;

typedef
  struct Subfont {
  char  *name;
  short n;    /* number of chars in subfont */
  uchar height;  /* height of image */
  char ascent;  /* top of image to baseline */
  Fontchar *info;  /* n+1 Fontchars */
  Image  *info;  /* of font */
  } Subfont;

The image fills the rectangle (0, 0, w, height), where w is the sum of the horizontal
extents (of non-zero pixels) for all characters. The pixels to be displayed for character c are in the
rectangle (i->x, i->top, (i+1)->x, i->bottom) where i is &subfont->info[c].
When a character is displayed at Point p in an image, the character rectangle is placed at
(p.x+i->left, p.y) and the next character of the string is displayed at (p.x+i->width, p.y). The baseline of the characters is ascent rows down from the top of the subfont image. The
info array has n+1 elements, one each for characters 0 to n–1 plus an additional entry so
the size of the last character can be calculated. Thus the width, w, of the Image associated with a
Subfont s is s->info[s->n].x.

A Font consists of an overall height and ascent and a collection of subfonts together with the
ranges of runes (see utf(6)) they represent. Fonts are described by the following structures.

typedef
  struct Cachefont {
  Rune  min;    /* value of 0th char in subfont */
typedef

struct Cacheinfo {
    ushort x;  /* left edge of bits */
    uchar width; /* width of baseline */
    schar left;  /* offset of baseline */
    Rune value; /* of char at this slot in cache */
    ushort age;
} Cacheinfo;

typedef

struct Cachesubf {
    ulong age;  /* for replacement */
    Cachefont *cf;  /* font info that owns us */
    Subfont *f;  /* attached subfont */
} Cachesubf;

typedef

struct Font {
    char *name;
    Display *display;
    short height;  /* max ht of image; interline space */
    short ascent;  /* top of image to baseline */
    short width;  /* widest so far; used in caching */
    short nsub;  /* number of subfonts */
    ulong age;  /* increasing counter; for LRU */
    int ncache;  /* size of cache */
    int nsubf;  /* size of subfont list */
    Cacheinfo *cache;
    Cachesubf *subf;
    Cachefont **sub;  /* as read from file */
    Image *cacheimage;
} Font;

The height and ascent fields of Font are described in graphics(2). Sub contains nsub pointers to Cachefonts. A Cachefont connects runes min through max, inclusive, to the subfont with file name name; it corresponds to a line of the file describing the font.

The characters are taken from the subfont starting at character number offset (usually zero) in the subfont, permitting selection of parts of subfonts. Thus the image for rune r is found in position r-min+offset of the subfont.

For each font, the library, with support from the graphics server, maintains a cache of subfonts and a cache of recently used character images. The subf and cache fields are used by the library to maintain these caches. The width of a font is the maximum of the horizontal extents of the characters in the cache. String draws a string by loading the cache and emitting a sequence of cache indices to draw. Cachechars guarantees the images for the characters pointed to by *s or *r (one of these must be nil in each call) are in the cache of f. It calls loadchar to put missing characters into the cache. Cachechars translates the character string into a set of cache indices which it loads into the array c, up to a maximum of n indices or the length of the string. Cachechars returns in c the number of cache indices emitted, updates *s to point to the next character to be processed, and sets *widp to the total width of the characters processed. Cachechars may return before the end of the string if it cannot proceed without destroying active data in the caches. If it needs to load a new subfont, it will fill *sfname with the name of the subfont it needs and return -1. It can return zero if it is unable to make progress because it cannot resize the caches.
Loadchar loads a character image into the character cache. Then it tells the graphics server to copy the character into position h in the character cache. If the current font width is smaller than the horizontal extent of the character being loaded, loadfont clears the cache and resets it to accept characters with the bigger width, unless noclr is set, in which case it just returns -1. If the character does not exist in the font at all, loadfont returns 0; if it is unable to load the character without destroying cached information, it returns -1, updating *sfname as described above. It returns 1 to indicate success.

The age fields record when subfonts and characters have been used. The font age is increased every time the font is used (agefont does this). A character or subfont age is set to the font age at each use. Thus, characters or subfonts with small ages are the best candidates for replacement when the cache is full.

SOURCE
/sys/src/libdraw

SEE ALSO
graphics(2), allocimage(2), draw(2), subfont(2), image(6), font(6)

DIAGNOSTICS
All of the functions use the graphics error function (see graphics(2)).
NAME
  chdir - change working directory

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  int chdir(char *dirname)

DESCRIPTION
  Chdir changes the working directory of the invoking process to dirname. The working directory is the starting point for evaluating file names that do not begin with / or #, as explained in intro(2). When Plan 9 boots, the initial process has / for its working directory.

SOURCE
  /sys/src/libc/9syscall

SEE ALSO
  intro(2)

DIAGNOSTICS
  Sets errstr.
NAME
cleanname - clean a path name

SYNOPSIS
#include <u.h>
#include <libc.h>

char* cleanname(char *filename)

DESCRIPTION
Cleanname takes a filename and by lexical processing only returns the shortest string that names
the same (possibly hypothetical) file. It eliminates multiple and trailing slashes, and it lexically
interprets . and .. directory components in the name. The string is overwritten in place.

The shortest string cleanname can return is two bytes: the null-terminated string ".". Therefore
filename must contain room for at least two bytes.

SOURCE
/sys/src/libc/port/cleanname.c

SEE ALSO

cleanname(1)
NAME
cmap2rgb, cmap2rgba, rgb2cmap – colors and color maps

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
int rgb2cmap(int red, int green, int blue)
int cmap2rgb(int col)
int cmap2rgba(int col)

DESCRIPTION
These routines convert between ‘true color’ red/green/blue triples and the Plan 9 color map. See color(6) for a description of RGBV, the standard color map.

Rgb2cmap takes a trio of color values, scaled from 0 (no intensity) to 255 (full intensity), and returns the index of the color in RGBV closest to that represented by those values.

Cmap2rgb decomposes the color of RGBV index col and returns a 24–bit integer with the low 8 bits representing the blue value, the next 8 representing green, and the next 8 representing red.

Cmap2rgba decomposes the color of RGBV index col and returns a 32–bit integer with the low 8 bits representing an alpha value, defined to be 255, and the next 8 representing blue, then green, then red, as for cmap2rgb shifted up 8 bits. This 32–bit representation is the format used by draw(2) and memdraw(2) library routines that take colors as arguments.

SOURCE
/sys/src/libdraw

SEE ALSO
graphics(2), allocimage(2), draw(2), image(6), color(6)
NAME
  complete - file name completion

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  #include <complete.h>

typedef struct CompletionCompletion;
struct Completion{
  uchar advance; /* whether forward progress has been made */
  uchar complete; /* whether the completion now represents a file or directory */
  char *string; /* the string to advance, suffixed " " or "/" for file or directory */
  int nmatch; /* number of files that matched */
  int nfile; /* number of files returned */
  char **filename; /* their names */
};

Completion* complete(char *dir, char *s);
void freecompletion(Completion *c);

DESCRIPTION
  The complete function implements file name completion. Given a directory dir and a string s, it returns an analysis of the file names in that directory that begin with the string s. The fields nmatch and nfile will be set to the number of files that match the prefix and filename will be filled in with their names. If the file named is a directory, a slash character will be appended to it.

  If no files match the string, nmatch will be zero, but complete will return the full set of files in the directory, with nfile set to their number.

  The flag advance reports whether the string s can be extended without changing the set of files that match. If true, string will be set to the extension; that is, the value of string may be appended to s by the caller to extend the embryonic file name unambiguously.

  The flag complete reports whether the extended file name uniquely identifies a file. If true, string will be suffixed with a blank, or a slash and a blank, depending on whether the resulting file name identifies a plain file or a directory.

  The freecompletion function frees a Completion structure and its contents.

  In rio(1) and acme(1), file name completion is triggered by a control-F character or an Insert character.

SOURCE
  /sys/src/libcomplete

SEE ALSO
  rio(1), acme(1)

DIAGNOSTICS
  The complete function returns a null pointer and sets errstr if the directory is unreadable or there is some other error.

BUGS
  The behavior of file name completion should be controlled by the plumber.
NAME
Control, Controlset, activate, closecontrol, closecontrolset, controlcalled, controlwire, createbox, createboxbox, createbutton, createcolumn, createentry, createkeyboard, createlabel, createmenu, creatoradiobutton, createrow, createscribble, createslider, createstack, createtab, createtext, createtextbutton, cterror, ctmalloc, ctirealloc, ctstrdup, ctlprint, deactivate, freectlfont, freectlimage, initcontrols, namectlfont, namectlimage, newcontrolset, resizecontrolset – interactive graphical controls

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <thread.h>
#include <keyboard.h>
#include <mouse.h>
#include <control.h>
typedef struct Control Control;
typedef struct Controlset Controlset;
struct Control
{
    char *name;
    Rectangle rect; /* area on screen */
    Rectangle size; /* min/max Dx, Dy (not a rect) */
    Channel *event; /* chan(char*) to client */
    Channel *data; /* chan(char*) to client */
    ...};
struct Controlset
{
    ...
    Channel *ctl;
    Channel *data;
    ...
    int clicktotype;
    ...};
void initcontrols(void)
Controlset*newcontrolset(Image *i, Channel *kc, Channel *mc, Channel *rc)
void closecontrolset(Controlset *cs)
int namectlfont(Font *font, char *name)
int freectlfont(char *name)
int namectlimage(Image *image, char *name)
int freectlimage(char *name)
Control*createbox(Controlset *cs, char *name)
Control*createboxbox(Controlset *cs, char *name)
Control*createbutton(Controlset *cs, char *name)
Control*createcolumn(Controlset*, char*)
Control*createentry(Controlset *cs, char *name)
Control*createkeyboard(Controlset *cs, char *name)
Control*createlabel(Controlset *cs, char *name)
Control*createmenu(Controlset *cs, char *name)
Control*creatoradiobutton(Controlset *cs, char *name)
Control*createrow(Controlset*, char*)
Control* createscribble(Controlset *cs, char *name)
Control* createslider(Controlset *cs, char *name)
Control* createtab(Controlset*, char*)
Control* createtext(Controlset *cs, char *name)
Control* createtextbutton(Controlset *cs, char *name)
void closecontrol(Control *c)
int ctlprint(Control*, char*, ...);
void ctlerror(char *fmt, ...)
Control* controlcalled(char *name)
void controlwire(Control *c, char *cvalue, Channel *ch)
void activate(Control *c)
void deactivate(Control *c)
void resizecontrolset(Controlset *cs)
void* ctlmalloc(uint n)
void* ctlrealloc(void *p, uint n)
char* ctlstrdup(char *s)
int ctldeletequits;

DESCRIPTION
This library provides a set of interactive controls for graphical displays: buttons, sliders, text entry boxes, and so on. It also provides aggregator Controls: boxes, columns, rows and stacks of Controls. A stack is a collection of co-located Controls, of which one is normally visible. A Controlset collects a group of Controls that share mouse and keyboard. Each Controlset has a separate thread of control that processes keyboard and mouse events as well as commands to be passed on to the Controls. Since each Controlset uses a thread, programs using the control library must be linked with the thread library, thread(2).

Controls are manipulated by reading and writing to the control channel, ctl, of their Controlset. Channels are defined in thread(2). Each Control has two output channels: Event delivers messages about actions within the control (such as a button press) and data delivers (if requested by an appropriate write to ctl) control-specific data such as the contents of a field.

The library provides a simple mechanism for automatic layout: the minimum and maximum sizes of each simple control can be specified. Boxbox, row, column and stack Controls then use these sizes to lay out their constituent Controls when called upon to do so. See the description of these grouping Controls for further details.

Message format
All messages are represented as UTF-8 text. Numbers are formatted in decimal, and strings are transmitted in the quoted form of quote(2).

Messages sent to a Controlset are of the form,

sender: destination verb [argument ...]

The sender (and the colon following it) may be ommitted. For example, the initial field of a text entry control called entry could be set by sending the message,

entry value 'Hello, world!'

to its Controlset's ctl file. This message contains the verb value and the single argument Hello, world!

To make it easy to write messages, the function chanprint (see thread(2)) can be used to print formatted text to a Controlset's channel.

The %q and %Q formats are convenient for properly quoting string arguments, as in

chanprint(e−>event, "value %q", "Don't touch!");
It is wise to use %q always instead of %s when sending messages, and avoid dealing with the quoting explicitly. In the other direction, tokenize (see getfields(2)) parses these messages and interprets the quotes correctly.

The destination of a message can be a named control, or a set of controls identified by name or type. The command

    'entry slider' show

(note the quotation) sends the 'show' command to the entry named entry and all controls of type slider. If there were a control whose name was slider that control would also be shown.

Note that we are still experimenting with destination names. One proposal is that a destination of the form "'name1 name2 ... type1 type2 ...' selects all controls of the named types in the control hierarchies (of columns, rows and stacks) whose names precede the types.

Messages sent by a control on its event channel are of the form

    sender: event

The sender is the name of the control sending the message; the event describes the event. Its format can often be controlled by setting the Control's format string. For example, when the user types a newline at a text entry Control named entry, the control sends the message

    entry: value 'Hello again!' on its event channel.

Initialization and Control sets

After initdraw (see graphics(2)) is called, the function initcontrols should be called to initialize the library. It calls quotefminstall to install the %q and %Q formats; see quote(2).

Each control is represented by a Control data structure and is associated with a Controlset that groups a set of controls sharing mouse, keyboard, and display. Most applications will need only one Controlset; only those with multiple windows or unusual configurations will need more than one. The function newcontrolset creates a Controlset. Its arguments are the image (usually a window) on which its controls will appear, typically the screen variable in the draw library, and three channels: kc, a channel of Runes from the keyboard; mc, a channel of Mouse structures from the mouse; and rc, a channel of int that indicates when the window has been resized. Any of the channels may be nil, in which case newcontrolset will call initkeyboard and/or initmouse (see keyboard(2) and mouse(2)) to initialize the keyboard and mouse and connect them to the control set. The mouse and resize channels must both be nil or both be non-nil.

The function closecontrolset frees all the controls in the control set and tears down all the associated threads. It does not close the mouse and keyboard.

The public elements of a Controlset are the flag clicktotype, and the ctl and data channels.

Clicktotype is zero by default. If it is set to non–zero, the controls in the set will acquire ‘focus’ by the click–to–type paradigm. Otherwise, focus is always given to the control under the mouse.

Commands for controls are sent through the Controlset’s ctl channel. One special command is recognized by the Controlset itself: Sending the string sync to the ctl channel causes the string to be echoed to the Controlset’s data channel when all commands up to the sync command have been processed. The string is allocated and must be freed (see malloc(2)). Synchronization is necessary between sending a command, for example, to resize all controls, and using their rect fields.

The function resizecontrolset must be provided by the user. When the associated window is resized, the library will call resizecontrolset with the affected Controlset; the function should reconnect to and redraw the window.

If all windows are organized in a hierarchy of boxboxes, columns, rows and stacks, and minimum and maximum sizes have already been supplied, only the top control needs to be resized (see the rect command below).

Fonts and images

Fonts and images must be given names so they may be referenced in messages. The functions namectlfont and namectlimage associate a unique name with the specified font or image. The association is removed by freectlfont and freectlimage. The font or image is not freed by these
functions, however.

The function `initcontrols` establishes name bindings for all the colors mentioned in `<draw.h>`, such as black, white, red, yellow, etc., as well as masks transparent and opaque. It also sets the name font to refer to the default font variable set up by `initdraw`.

**Creation**

Each type of control has an associated creation function: `createbutton`, `createentry`, etc., whose arguments are the Controlset to attach it to and a globally unique name for it. A control may be destroyed by calling `closecontrol`.

The function `controlcalled` returns a pointer to the Control with the given `name`, or nil if no such control exists.

**Configuration**

After a control is created, it must be configured using the control-specific commands documented below. Commands are sent to the `ctl` channel of the Controlset. Multiple commands may be sent in a single message; newline characters separate commands. For an example, see the implementation of `resizecontrolset` in the EXAMPLES section. Note that newline is a separator, not a terminator; the final command does not need a newline.

Messages sent to the `ctl` channel are delivered to all controls that match the `destination` field. This field is a set of names separated by spaces, tabs or newlines. A control matches the destination if its name or its type is among the set.

The recipient of a message ignores the initial `sender`: field of the message, if present, making it possible to send messages generated on an event channel directly to another control’s `ctl` channel.

**Activation**

When they are created, controls are disabled: they do not respond to user input. Not all controls need to be responsive; for example, labels are static and a text display might show a log of messages but not be useful to edit. But buttons, entry boxes, and other text displays should be active.

To enable a control, call the `activate` function, which specifies that the Control `c` should respond to mouse and keyboard events; `deactivate` turns it off again.

Controls can be either `revealed` (default) or `hidden`. When a control is hidden, it will not receive mouse or keyboard events and state changes or `show` commands will be ignored until the control is once again `revealed`. Control hiding is particularly useful when different controls are overlayed, revealing only the ‘top’ one.

The function `controlwire` permits rearrangement of the channels associated with a Control. The channel `cname` (one of "data" or "event") of Control `c` is reassigned to the channel `ch`. There are several uses for this operation: one may reassign all the event channels to a single channel, in effect multiplexing all the events onto a single channel; or connect the event channel of a slider to the `ctl` channel for delivery to a text display (after setting the format for the slider’s messages to name the destination control and the appropriate syntax for the rest of the command) to let the slider act as a scroll bar for the text without rerouting the messages explicitly.

**Controls**

The following sections document the individual controls in alphabetical order. The layout of each section is a brief description of the control’s behavior, followed by the messages it sends on event, followed by the messages it accepts via the `ctl` channel. The event messages are triggered only by mouse or keyboard action; messages to the `ctl` file do not cause events to be generated.

All controls accept the following messages:

- `rect minx miny maxx maxy`
  
  Set the bounding rectangle for the control on the display. The syntax generated by the `%R` print format of the draw library is also acceptable for the coordinates.

- `size [minΔx minΔy maxx maxΔy]`

  Set the minimum and maximum size for automatic layout in `columns`, `rows` and `stacks`. Without its four arguments, this command is ignored by primitive controls and used by grouping controls to calculate their minimum and maximum sizes by examining those of their constituent members. If all primitive controls have been assigned a size, a
single size request addressed to the top of a layout hierarchy will assign sizes to all
grouping Controls.

hide  Disable drawing of the control and ignore mouse and keyboard events until the control
is once again revealed. Grouping Controls (column, row, and stack) pass the
request down to their constituent Controls.

reveal This is the opposite of hide: the Control is displayed and mouse and keyboard
operations resume. Grouping Controls (column, row, and stack) pass the request
down to their constituent Controls. The reveal command for stacks takes an
optional argument naming the Control to be revealed; all other Controls will be
hidden.

show  Display the Control on its screen if not hidden. Some actions will also cause the
Controls to show themselves automatically (but never when the control is hid-
den). Grouping Controls (column, row, and stack) pass the request down to their
constituent Controls.

Many messages are common between multiple Controls. Such messages are described in detail
here to avoid repetition. In the individual descriptions, only the syntax is presented.

align n  Specify the alignment of (some part of) the Control’s display within its rectan-
gle. For textual controls, the alignment specifies where the text should
appear. For multiline text, the alignment refers to each line within its box, and
only the horizontal part is honored. For other Controls, the alignment affects
the appearance of the display in a reasonable way. The valid alignments are
words with obvious interpretations: upperleft, uppercenter, upperright, centerleft, center, centerright, lowerleft,
lowercenter, and lowerright.

border n  Inset the Control (or separate constituent Controls in box, column and
row Controls after the next rect command) within its rectangle by n pixels,
default zero.

bordercolor name  Paint the border of the control with the named color, default black.

focus n  The Control now has (if n is non-zero) or does not have (if n is zero) focus.
Most Controls ignore the message; there are plans to make them react.

format fmt  Set the format of ‘value’ messages sent on the event channel. By default, the
format is "%q: value %q" for string-valued Controls, "%q: value
%d" for integer-valued Controls such as buttons, and "%q: value
0x%x" for the keyboard and scribble Controls. The %q prints the name of the
Control; the rest the value. Any supplied format string must be type-

image name  Many controls set a background image or color for display. The image message

light name  sets the image. The mask and light images together specify how the
control shows it is enabled: the light is printed through the mask when
the state is ‘on’ or ‘pressed’. Otherwise, the image appears unmodified. The
default image is white; mask opaque; light yellow.

font name  These commands set the font and color for displaying text. The defaults are the
textcolor name  default font set up by the draw library, and black.

value v  Set the value of the Control. Textual images accept an arbitrary string; others
an integral value.

Box
A box is a trivial control that does nothing more than pass keyboard, mouse, and focus messages
back on its event channel. Keyboard characters are sent in the format

boxname: key 0xnn

where nn is the hexadecimal value of the character. Mouse messages are sent in the format

boxname: mouse [x y] but msec
where \( x, y, \) but, and \( msec \) are the various fields of the Mouse structure. The focus message is just

\[
\text{boxname: focus n}
\]

where \( n \) is 0 if the box has lost focus, 1 if it has acquired it.

The box displays within its rectangle an image, under mask, with specified alignment. The control messages it accepts are:

align \( a \) Controls the placement of the image in the rectangle (unimplemented).
border \( b \)
bordercolor \( name \)
focus \( n \)
hide
image \( name \)
rect \( \text{minx miny maxx maxy} \) reveal
show
size \( \text{min\Delta x min\Delta y max\Delta x max\Delta y} \)

**Boxbox**

A boxbox allows a set of controls (‘boxes’) to be displayed in rows and columns within the rectangle of the boxbox. The maximum of the minimum heights of the constituent controls determines the number of rows to be displayed. The number of columns is the minimum that allows all Controls to be displayed. This aggregator works well for collections of buttons, labels, or textbuttons that all have a fixed height.

add \( name ... \) adds the named control to the box of controls. The display order is determined by the order of adding. The first named control is top left, the second goes below it, etc. It is possible to add one control to multiple grouping controls but the layout of the result will be quite unpredictable.

border \( width \)
bordercolor \( color \)
hide This command is passed on to the member controls.
image \( color \) Background color displayed between member controls.
reveal This command is passed on to the member controls.
separation \( width \) Set the separation between member controls to \( n \) pixels.
rect \( \text{minx miny maxx maxy} \) The member controls are laid out within the given rectangle according to the minimum and maximum sizes given. If the rectangle is not large enough for the minimum a fatal error is currently generated. If the controls at their maximum size are not big enough to fit, they are top–left justified at their maximum size in the space given. Otherwise, controls will get their minimum size and be enlarged proportional to the extra size given by the maximum until they fit given rectangle. The members are separated by borders of the width established by \( \text{borderwidth} \).
remove \( name \) Remove the named control from the box.
show This command is passed on to the member controls. Show also (re)displays background and borders.
size \( \text{min\Delta x min\Delta y max\Delta x max\Delta y} \)

**Button**

A button is a simple control that toggles its state when mouse button 1 is pressed on its rectangle. Each state change triggers an event message:

\[
\text{buttonname: value n}
\]

where \( n \) encodes the mouse buttons used to make the selection.

The button displays an image (which may of course be a simple color) and illuminates in the standard way when it is ‘on’. The control messages it accepts are:

align \( a \) Controls the placement of the image in the rectangle (unimplemented).

326
border b
bordercolor name
focus n
format fmt
hide
image name
light name
mask name
rect minx miny maxx maxy
reveal
show
size minx miny maxx maxy
value n Set the button to 'on' (if n is non-zero) or 'off' (if n is zero).

Column
A column is a grouping control which lays out its members vertically, from top to bottom. Currently, columns ignore mouse and keyboard events, but there are plans to allow dragging the borders (when they have non-zero width) between constituent members.

add name ... adds the named control to the column of controls. The vertical order is determined by the order of adding. The first named control goes at the top. It is possible to add one control to multiple grouping controls but the layout of the result will be quite unpredictable.

border width Set the border between members to the width given.
bordercolor color
hide
image color Background color displayed between member controls.
reveal
separation width
show These three commands are passed on to the member controls. Show also (re)displays the borders between members.

rect minx miny maxx maxy
The member controls are layed out within the given rectangle according to the minimum and maximum sizes given. If the rectangle is not large enough for the minimum a fatal error is currently generated. However, see the example at the end of this man page. If the controls at their maximum size are not big enough to fit, they are centered at their maximum size in the space given. Otherwise, controls will get their minimum size and be enlarged proportional to the extra size given by the maximum until they fit given rectangle. The members are separated by borders of the width established by borderwidth.

remove name Remove the named control from the column.
size [ minΔx minΔy maxΔx maxΔy ]
Without arguments, this command computes the minimum and maximum size of a column by adding the minimum and maximum heights to set minΔy and maxΔy, and it finds the largest minimum and maximum widths to set minΔx and maxΔx. When called with arguments, it simply sets the minimum and maximum sizes to those given.

Entry
The entry control manages a single line of editable text. When the user hits a carriage return anywhere in the text, the control generates the event message,

   entryname: value s

with s the complete text of the entry box.

The cursor can be moved by clicking button 1; at the moment, there is no way to select characters, only a typing position. Some control characters have special actions: control–H (backspace) deletes the character before the cursor; control–U clears the line; and control–V pastes the snarf buffer at the typing position. Most important, carriage return sends the text to the event channel.

To enter passwords and other secret text without displaying the contents, set the font to one in which all characters are the same. The easiest way to do this is to make a font containing only one
character, at position 0 (NUL), since that position is used to render all characters not otherwise defined in the font (see draw(2)). The file /lib/font/bit/lucm/passwd.9.font defines such a font.

The control messages the entry control accepts are:

align a Controls the placement of the text in the rectangle.
border b
bordercolor name
data After receiving this message, the entry will send its value to its data channel as an unadorned, unquoted string.
focus n When it receives focus, the entry box displays a typing cursor. When it does not have focus, the cursor is not displayed.
font name format fmt
hide
image name
rect minx miny maxx maxy
reveal
show
size minΔx minΔy maxΔx maxΔy
textcolor name
value s Set the string displayed in the entry box.

Keyboard
The keyboard control implements a simulated keyboard useful on palmtop devices. Keystrokes, generated by mouse button 1 on the simulated keys, are sent as event messages:

keyboardname: value 0xnn

where nn is the hexadecimal Unicode value of the character. Shift, control, and caps lock are handled by the keyboard control itself; shift and control affect only the next regular keystroke. The Alt key is unimplemented; it will become equivalent to the standard Plan 9 key for synthesizing non-ASCII characters.

There are two special keys, Scrib and Menu, which return values 0x10000 and 0x10001.

The image, mask, light rules are used to indicate that a key is pressed, but to aid clumsy fingers the keystroke is not generated until the key is released, so it is possible to slide the pointer to a different key to correct for bad aim.

The control messages the keyboard accepts are:

border b
bordercolor name
focus n
font name1 name2
Sets the font for the keys. If only one font is named, it is used for all keys. If two are named, the second is used for key caps with special names such as Shift and Enter. (Good choices on the Bitsy are /lib/font/bit/lucidasans/boldlatin1.6.font for the first and /lib/font/bit/lucidasans/unicode.6.font for the second argument.) If neither is specified, both will be set to the default global font.

format fmt
hide
image name
light name
mask name
rect minx miny maxx maxy
reveal
show
size minΔx minΔy maxΔx maxΔy

Label
A label is like a textbutton (q.v.) that does not react, but whose value is the text it displays. The
control messages it accepts are:

align a Controls the placement of the image in the rectangle.
border b
bordercolor name
focus n
font name
hide
image name
rect minx miny maxx maxy
reveal
show
size minx miny maxx maxy
textcolor name
value s The value is a string that can be modified only by sending this message to the ctl file.

Menu
A menu is a pop-up window containing a set of textual selections. When a selection is made, it removes itself from the screen and reports the selection by value:

menuname: value n

If no selection is made, no message is reported. Because it creates a window, programs using a menu must have their screen variable (see graphics(2) and window(2)) set up to be refreshed properly. The easiest way to do this is to call getwindow with refresh argument RefBackup (see graphics(2)); most programs use RefNone.

The control messages accepted by a menu are:

add text Add a line of text to the end of the menu.
align a Controls the left-right placement of the text in its rectangle.
border b
bordercolor name
focus n
font name
format fmt
hide
image name
rect minx miny maxx maxy
reveal
size minx miny maxx maxy

Only the origin of the rectangle is significant; menus calculate the appropriate size.
selectcolor name
Set the color in which to highlight selected lines; default yellow.
selecttextcolor name
Set the color in which to draw the text in selected lines; default black.
show Display the menu. Not usually needed unless the menu is changed while visible; use window instead.
window window n With no arguments, toggle the menu’s visibility; otherwise make it visible (1) or invisible (0). When the selection is made, the menu will remove its window automatically.

Radiobutton
The radiobutton assembles a group of buttons or textbuttons into a single control with a numeric value. Its value is -1 if none of the constituent buttons is pressed; otherwise it is the index, starting at zero, of the button that is pressed. Only one button may be pressed; the radiobutton manipulates its buttons to guarantee this. State changes trigger an event message:

radiobuttonname: value n

Buttons are added to the radio button using the add message; there is no way to remove them, although they may be turned off independently using deactivate. The index reported in the value is defined by the order in which the buttons are added. The constituent buttons should be
configured and layed out in the usual way; the rectangle of the radiobutton is used only to 'catch' mouse events and should almost always correspond to the bounding box of the constituent buttons. In other words, the geometry is not maintained automatically.

The control messages the radiobutton accepts are:

add name  Add the control with the specified name to the radiobutton.
focus n  
format fmt  
hide  
rect minx miny maxx maxy  
reveal  
size minx miny maxx maxy  
show  
value n  

Row

A row groups a number of member controls left to right in a rectangle. Rows behave exactly like columns with the roles of x and y interchanged.

The control messages it accepts are:

add name ...  
border width  
bordercolor color  
hide  
image color  
rect minx miny maxx maxy  
remove name  
reveal  
separation width  
show  
size [min\Delta x min\Delta y max\Delta x max\Delta y]  

Scribble

The scribble control provides a region in which strokes drawn with mouse button 1 are interpreted as characters in the manner of scribble(2). In most respects, including the format of its event messages, it is equivalent to a keyboard control.

The control messages it accepts are:

align a  Controls the placement of the image in the rectangle (unimplemented).
border b  
bordercolor name  
focus n  
font name  Used to display the indicia.
hide  
image name  
linelcolor name  The color in which to draw the strokes; default black.
rect minx miny maxx maxy  
reveal  
size minx miny maxx maxy  
show  

Stack

A stack groups a number of member controls in the same shared rectangle. Only one of these controls will be visible (revealed), the others are hidden.

The control messages it accepts are:

hide  
rect minx miny maxx maxy  
remove name  
reveal[n] Without argument, reveal is the opposite of hide: it makes its selected control visible after it was hidden. With an argument, it makes the n'th added control visible, hiding all others.
show
size [ min\Delta x min\Delta y max\Delta x max\Delta y ]

Without argument, size computes the maximum of the minimum and maximum sizes of its constituent controls. With arguments, it sets the size to the given values.

Slider
A slider controls an integer value by dragging the mouse with a button. Configured appropriately, it can serve as a scroll bar with the standard Plan 9 behavior. When the value changes, an event message is sent:

slidername: value n

The slider is a good candidate for connecting to another control by setting its format and rewiring its event channel to the other's ctl channel.

The geometry of the slider is defined by three numbers: max is a number representing the range of the slider; vis is a number representing how much of what is being controlled is visible; and value is a number representing the value of the slider within its range. For example, if the slider is managing a textual display of 1000 lines, with 18 visible, and the first visible line (numbered starting form 0) is 304, max will be 1000, vis will be 18, and value will be 304. The indicator is the visual representation of the vis portion of the controlled object.

The control messages the slider accepts are:

absolute n If n is zero, the slider behaves like a Plan 9 scroll bar: button 2 sets absolute position, button 1 decreases the value, and button 3 increases it. If n is nonzero, all buttons behave like button 2, setting the absolute value.

border b
bordercolor name

clamp end n The end is either the word high or low; n sets whether that end is clamped or not. If it is clamped, that end of the indicator is always at its supremum. A standard scroll bar has neither end clamped; a volume slider would have its low end clamped. If the low end is clamped, the value of the slider is represented by the high end of the indicator; otherwise it is represented by the low end.

focus n
format fmt
hide
image name
indicatorcolor name

max n Set the maximum value of the range covered by the slider.

orient dir The string dir begins either hor or ver to specify the orientation of the slider. The default is vertical. The value always increases to the right for horizontal sliders and downwards for vertical sliders.

rect minx miny maxx maxy
reveal

size minx miny maxx maxy
show
value n Set the visible area shown by the indicator.

Tab
A tab control combines radiobottuns with a stack of windows giving the appearance of tabbed controls. Currently, the tabs are positioned at the top of the stack. The radiobutton consists of textbuttons, the stack can be composed of any type of control.

Control messages are

add button control button control ...

Adds a button to the radiobutton, and an associated control to the stack. Buttons and controls are numbered in the order of addition. There is no remove operation.

border b
bordercolor color
focus n
format fmt When a format string is defined, the tab control reports which tab is selected
image color Using the format string (which must print a char* and an int).
separation n Color between member controls.
rect n n n n Spacing between buttons in the radiobutton and between the row of buttons and
hide the stack below it.
reveal size n n n n
show value n Value must be an integer indicating which tab to bring to the top.

Text
A text control presents a set of lines of text. The text cannot be edited with the keyboard, but can
be changed by control messages. (A more interactive text control will be created eventually.) The
mouse can be used to select lines of text. The only event message reports a state change in the
selection of a line:

textname: select n s

states that line n has changed its selection state to s, either zero (unselected) or non-zero
(selected). The non–zero value encodes the mouse buttons that were down when the selection
occurred.

The control messages the text control accepts are:
accumulate s
accumulate n s
add s
add n s With one argument, append the string s as a new last line of the control; if n
is specified, add the line before the current line n, making the new line number n. The lines are zero indexed and n can be no greater than the current
number of lines. Add refreshes the display, but accumulate does not, to
avoid n–squared behavior when assembling a piece of text.
align a Controls the placement of each line of text left–to–right in its rectangle. Vertically, lines are tightly packed with separation set by the font’s interline
spacing.

border b
bordercolor name
clear Delete all text.
delete n Delete line n.
focus n
font name
image name
rect minx miny maxx maxy
replace n s Replace line n by the string s.
reveal
scroll n If n is non–zero, the text will automatically scroll so the last line is always vis-
select n m Set the selection state of line n to m.
selectcolor name able when new text is added.
Set the color in which to highlight selected lines; default yellow.
selectmode s The string s is either single or multi. If single, the default, only one
line may be selected at a time; when a line is selected, other lines are un-
selected. If multi, the selection state of individual lines can be toggled inde-
dependently.
size minx miny maxx maxy
show
textcolor name
EXAMPLES

This example creates two entry boxes, top and bot, and copies the contents of one to the other whenever a newline is typed.

```c
#include <u.h>
#include <libc.h>
#include <thread.h>
#include <draw.h>
#include <mouse.h>
#include <keyboard.h>
#include <control.h>

Controlset *cs;
```

**Textbutton**

A textbutton is a textual variant of a plain button. Each state change triggers an event message:

```
textbuttonname: value n
```

where *n* encodes the mouse buttons used to make the selection.

Like a regular button, the value of a textbutton is an integer; the *text* is the string that appears in the button. It uses the image, light, mask method of indicating its state; moreover, the color of the text can be set to change when the button is pressed. The control messages it accepts are:

- **align** *a*  Controls the placement of the text in the rectangle.
- **border** *b*
- **bordercolor** *name*
- **focus** *n*
- **font** *name*
- **format** *fmt*
- **hide**
- **image** *name*
- **light** *name*
- **mask** *name*
- **pressedtextcolor** *name*
  - Set the color in which to display text when the textbutton is pressed.
- **rect** *minx miny maxx maxy*
- **reveal**
- **size** *minx miny maxx maxy*
- **show**
- **text** *s*  Set the text displayed in the button.
- **textcolor** *name*
- **value** *n*  Set the button to ‘on’ (if *n* is non-zero) or ‘off’ (if *n* is zero).

**Helper functions**

The function `ctlerror` is called when the library encounters an error. It prints the formatted message and exits the program.

The functions `ctlmalloc`, `ctlrealloc`, `ctlstrdup`, and `ctlunestrdup` are packagings of the corresponding C library functions. They call `ctlerror` if they fail to allocate memory, and `ctlmalloc` zeros the memory it returns.

Finally, for debugging, if the global variable `ctldeletequits` is set to a non-zero value, typing a DEL will cause the program to call

```
ctlerror("delete");
```

**Caveat**

This library is very new and is still missing a number of important features. The details are all subject to change. Another level of library that handles geometry and has sensible default appearances for the controls would be useful.

One unusual design goal of this library was to make the controls themselves easy to implement. The reader is encouraged to create new controls by adapting the source to existing ones.

**topline** *n*  Scroll the text so the top visible line is number *n*.

**value** *s*  Delete all the text in the control and then add the single line *s*. 
int ctldeletequits = 1;
void
resizecontrolset(Controlset*)
{
    int i;
    Rectangle r, r1, r2;
    if(getwindow(display, Refnone) < 0)
        sysfatal("resize failed: %r");
    r = insetrect(screen->r, 10);
    r1 = r;
    r2 = r;
    r1.max.y = r1.min.y+1+font->height+1;
    r2.min.y = r2.max.y+10;
    r2.max.y = r2.min.y+1+font->height+1;
    chanprint(cs->ctl, "top rect %R\ntop show", r1);
    chanprint(cs->ctl, "bot rect %R\nbot show", r2);
}
void
threadmain(int argc, char *argv[])
{
    char *s, *args[3];
    Channel *c;
    Control *top, *bot;
    int n;
    initdraw(0, 0, "example");
    initcontrols();
    cs = newcontrolset(screen, nil, nil, nil);
    cs->clicktotype = 1;
    top = createentry(cs, "top");
    chanprint(cs->ctl, "top image paleyellow");
    chanprint(cs->ctl, "top border 1");
    bot = createentry(cs, "bot");
    chanprint(cs->ctl, "bot image paleyellow");
    chanprint(cs->ctl, "bot border 1");
    c = chancreate(sizeof(char*), 0);
    controlwire(top, "event", c);
    controlwire(bot, "event", c);
    activate(top);
    activate(bot);
    resizecontrolset(cs);
    for(;;){
        s = recvp(c);
        n = tokenize(s, args, nelem(args));
        if(n==3 && strcmp(args[1], "value") == 0){
            if(strcmp(args[0], "top:") == 0)
                chanprint(cs->ctl, "bot value %q", args[2]);
            else
                chanprint(cs->ctl, "top value %q", args[2]);
        }
    }
    threadexitsall(nil);
}

A richer variant couples a text entry box to a slider. Since the value of a slider is its numerical setting, as a decimal number, all that needs changing is the setup of bot:
bot = createslider(cs, "bot");
chanprint(cs->ctl, "bot border 1");
chanprint(cs->ctl, "bot image paleyellow");
chanprint(cs->ctl, "bot indicatorcolor red");
chanprint(cs->ctl, "bot max 100");
chanprint(cs->ctl, "bot clamp low 1");
chanprint(cs->ctl, "bot orient horizontal");

The rest is the same. Of course, the value of the entry box is only meaningful to the slider if it is also a decimal number.

Finally, we can avoid processing events altogether by cross-coupling the controls. Replace the rest of threadmain with this:

chanprint(cs->ctl, "bot format %q", "%q: top value %q");
chanprint(cs->ctl, "top format %q", "%q: bot value %q");
controlwire(top, "event", cs->ctl);
controlwire(bot, "event", cs->ctl);
activate(top);
activate(bot);
resizecontrolset(cs);
for(;;)
    yield();
threadexitsall(nil);

SOURCE
/sys/src/libcontrol

SEE ALSO
draw(2), frame(2), graphics(2), quote(2), thread(2)

BUGS
The library is strict about matters of formatting, argument count in messages, etc., and calls ctlerror in situations where it may be fine to ignore the error and continue.
NAME
cputime, times, cycles – cpu time in this process and children

SYNOPSIS
#include <u.h>
#include <libc.h>
int times(long t[4])
double cputime(void)
void cycles(vlong *cyclep)

DESCRIPTION
If t is non–null, times fills it in with the number of milliseconds spent in user code, system calls, child processes in user code, and child processes in system calls. Cputime returns the sum of those same times, converted to seconds. Times returns the elapsed real time, in milliseconds, that the process has been running.

These functions read /dev/cputime, opening that file when they are first called.

Cycles reads the processor’s timestamp counter of cycles since reset, if any, and stores it via cyclep. Currently supported architectures are 386, amd64, and power; on all others, cycles will store zero.

SOURCE
/sys/src/libc/9sys
/sys/src/libc/*/cycles.[cs]

SEE ALSO
exec(2), cons(3)

BUGS
Only 386 processors starting with the Pentium have timestamp counters; calling cycles on earlier processors may execute an illegal instruction.
NAME
cime, localtime, gmtime, asctime, tm2sec, timezone - convert date and time

SYNOPSIS
#include <u.h>
#include <libc.h>
char* cime(long clock)
Tm* localtime(long clock)
Tm* gmtime(long clock)
char* asctime(Tm *tm)
l.long tm2sec(Tm *tm)
/env/timezone

DESCRIPTION
ctime converts a time clock such as returned by time(2) into ASCII (sic) and returns a pointer to a
30-byte string in the following form. All the fields have constant width.

 Wed Aug 5 01:07:47 EST 1973

Localtime and gmtime return pointers to structures containing the broken-down time. localtime
corrects for the time zone and possible daylight savings time; gmtime converts directly to GMT.
Asctime converts a broken-down time to ASCII and returns a pointer to a 30-byte string.

typedef
struct {
  int sec; /* seconds (range 0..59) */
  int min; /* minutes (0..59) */
  int hour; /* hours (0..23) */
  int mday; /* day of the month (1..31) */
  int mon; /* month of the year (0..11) */
  int year; /* year A.D. - 1900 */
  int wday; /* day of week (0..6, Sunday = 0) */
  int yday; /* day of year (0..365) */
  char zone[4]; /* time zone name */
  int tzoff; /* time zone delta from GMT */
} Tm;

Tm2sec converts a broken-down time to seconds since the start of the epoch. It ignores wday,
and assumes the local time zone if zone is not GMT.

When local time is first requested, the program consults the timezone environment variable to
determine the time zone and converts accordingly. (This variable is set at system boot time by
init(8).) The timezone variable contains the normal time zone name and its difference from GMT
in seconds followed by an alternate (daylight) time zone name and its difference followed by a
newline. The remainder is a list of pairs of times (seconds past the start of 1970, in the first time
zone) when the alternate time zone applies. For example:

EST -18000 EDT -14400
9943200 25664400 41392800 57718800 ...

Greenwich Mean Time is represented by

GMT 0

SOURCE
/sys/src/libc/9sys

SEE ALSO
date(1), time(2), init(8)

BUGS
The return values point to static data whose content is overwritten by each call.
Daylight Savings Time is “normal” in the Southern hemisphere.
These routines are not equipped to handle non-ASCII text, and are provincial anyway.
NAME
isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii,
toascii, _toupper, _tolower, toupper, tolower – ASCII character classification

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <ctype.h>
isalpha(c)  isgraph(c)
isupper(c)  iscntrl(c)
islower(c)  isasciic(c)
isdigit(c)  _toupper(c)
isxdigit(c)  _tolower(c)
isalnum(c)  toupper(c)
isspace(c)  tolower(c)
isprint(c)  toascii(c)
isprint(c)

DESCRIPTION
These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. Isscii is defined on all integer values; the rest are defined only where isscii is true and on the single non-ASCII value EOF; see fopen(2).
isalpha c is a letter, a–z or A–Z
isupper c is an upper case letter, A–Z
islower c is a lower case letter, a–z
isdigit c is a digit, 0–9
isxdigit c is a hexadecimal digit, 0–9 or a-f or A–F
isalnum c is an alphanumeric character, a–z or A–Z or 0–9
isspace c is a space, horizontal tab, newline, vertical tab, formfeed, or carriage return (0x20, 0x9, 0xA, 0xB, 0xC, 0xD)
ispunct c is a punctuation character (one of '!"#$%&'()*+,-./:;<=>?@[\]^_'{|}~)
isprint c is a printing character, 0x20 (space) through 0x7E (tilde)
isgraph c is a visible printing character, 0x21 (exclamation) through 0x7E (tilde)
iscntrl c is a delete character, 0x7F, or ordinary control character, 0x0 through 0x1F
isascii c is an ASCII character, 0x0 through 0x7F
Toascii is not a classification macro; it converts its argument to ASCII range by anding with 0x7F.
If c is an upper case letter, tolower returns the lower case version of the character; otherwise it returns the original character. Toupper is similar, returning the upper case version of a character or the original character. Ttolower and toupper are functions; _tolower and _toupper are corresponding macros which should only be used when it is known that the argument is upper case or lower case, respectively.

SOURCE
/sys/include/ctype.h  for the macros.
/sys/src/libc/port/ctype.c  for the tables.

SEE ALSO
isalpharune(2)

BUGS
These macros are ASCII–centric.
NAME

ciscrtrace, risctrace, ciscframe, riscframe, localaddr, symoff, fpformat, beieee80ftos, beieeesftos, beieedftos, leieee80ftos, leieesftos, leieedftos, ieeesftos, ieeedftos - machine-independent debugger functions

SYNOPSIS

#include <u.h>
#include <libc.h>
#include <bio.h>
#include <mach.h>

int ciscrtrace(Map *map, ulong pc, ulong sp, ulong link,
               Tracer trace)
int risctrace(Map *map, ulong pc, ulong sp, ulong link,
               Tracer trace)
ulong ciscframe(Map *map, ulong addr, ulong pc, ulong sp,
                ulong link)
ulong riscframe(Map *map, ulong addr, ulong pc, ulong sp,
                ulong link)
int localaddr(Map *map, char *fn, char *var, long *ret,
             Rgetter rget)
int symoff(char *buf, int n, long addr, int type)
int fpformat(Map *map, Reglist *rp, char *buf, int n, int code)
int beieee80ftos(char *buf, int n, void *fp)
int beieeesftos(char *buf, int n, void *fp)
int beieedftos(char *buf, int n, void *fp)
int leieee80ftos(char *buf, int n, void *fp)
int leieesftos(char *buf, int n, void *fp)
int leieedftos(char *buf, int n, void *fp)
int ieeesftos(char *buf, int n, ulong f)
int ieeedftos(char *buf, int n, ulong high, ulong low)

extern Machdata *machdata;

DESCRIPTION

These functions provide machine-independent implementations of common debugger functions. Many of the functions assume that global variables mach and machdata point to the Mach and Machdata data structures describing the target architecture. The former contains machine parameters and a description of the register set; it is usually set by invoking crackhdr (see mach(2)) to interpret the header of an executable. The Machdata structure is primarily a jump table specifying functions appropriate for processing an executable image for a given architecture. Each application is responsible for setting machdata to the address of the Machdata structure for the target architecture. Many of the functions described here are not called directly; instead, they are invoked indirectly through the Machdata jump table.

These functions must retrieve data and register contents from an executing image. The Map (see mach(2)) data structure supports the consistent retrieval of data, but no uniform access mechanism exists for registers. The application passes the address of a register retrieval function as an argument to those functions requiring register values. This function, called an Rgetter, is of the form

    ulong rget(Map *map, char *name);

It returns the contents of a register when given the address of a Map associated with an executing image and the name of the register.
Cisctrace and risctrace unwind the stack for up to 40 levels or until the frame for main is found. They return the count of the number of levels unwound. These functions process stacks conforming to the generic compiler model for RISC and CISC architectures, respectively. Map is the address of a Map data structure associated with the image of an executing process. Sp, pc and link are starting values for the stack pointer, program counter, and link register from which the unwinding is to take place. Normally, they are the current contents of the appropriate registers but they can be any values defining a legitimate process context, for example, an alternate stack in a multi-threaded process. Trace is the address of an application-supplied function to be called on each iteration as the frame unwinds. The prototype of this function is:

```c
void tracer(Map *map, ulong pc, ulong fp, Symbol *s);
```

where Map is the Map pointer passed to cisctrace or risctrace and pc and fp are the program counter and frame pointer. S is the address of a Symbol structure, as defined in symbol(2), containing the symbol table information for the function owning the frame (i.e., the function that caused the frame to be instantiated).

Ciscframe and riscframe calculate the frame pointer associated with a function. They are suitable for programs conforming to the CISC and RISC stack models. Map is the address of a Map associated with the memory image of an executing process. Addr is the entry point of the desired function. Pc, sp and link are the program counter, stack pointer and link register of an execution context. As with the stack trace functions, these can be the current values of the registers or any legitimate execution context. The value of the frame pointer is returned. A return value of zero indicates an error.

Localaddr fills the location pointed to by ret with the address of a local variable. Map is the address of a Map associated with an executing memory image. Fn and var are pointers to the names of the function and variable of interest. Rget is the address of a register retrieval function. If both fn and var are non-zero, the frame for function fn is calculated and the address of the automatic or argument named var in that frame is returned. If var is zero, the address of the frame for function fn is returned. In all cases, the frame for the function named fn must be instantiated somewhere on the current stack. If there are multiple frames for the function (that is, if it is recursive), the most recent frame is used. The search starts from the context defined by the current value of the program counter and stack pointer. If a valid address is found, localaddr returns 1. A negative return indicates an error in resolving the address.

Symoff converts a virtual address to a symbolic reference. The string containing that reference is of the form ‘name+offset’, where ‘name’ is the name of the nearest symbol with an address less than or equal to the target address and ‘offset’ is the hexadecimal offset beyond that symbol. If ‘offset’ is zero, only the name of the symbol is printed. If no symbol is found within 4,096 bytes of the address, the address is formatted as a hexadecimal address. Buf is the address of a buffer of n characters to receive the formatted string. Addr is the address to be converted. Type is the type code of the search space: CTEXT, CDATA, or CANY. Symoff returns the length of the formatted string contained in buf.

Fpformat converts the contents of a floating point register to a string. Map is the address of a Map associated with an executing process. Rp is the address of a Reglist data structure describing the desired register. Buf is the address of a buffer of n characters to hold the resulting string. Code must be either E or F, selecting double or single precision, respectively. If code is E, the contents of the specified register and the following register are interpreted as a double precision floating point number; this is only meaningful for architectures that implement double precision floats by combining adjacent single precision registers. For code F, the specified register is formatted as a single precision float. Fpformat returns 1 if the number is successfully converted or -1 in the case of an error.

Beieee80ftos, beieeesftos and beieeedftos convert big-endian 80-bit extended, 32-bit single precision, and 64-bit double precision floating point values to a string. Leieee80ftos, leieeesftos, and leieeedftos are the little-endian counterparts. Buf is the address of a buffer of n characters to receive the formatted string. Fp is the address of the floating point value to be converted. These functions return the length of the resulting string.

Ieieesftos converts the 32-bit single precision floating point value f, to a string in buf, a buffer of n bytes. It returns the length of the resulting string.
ieleedftos converts a 64-bit double precision floating point value to a character string. Buf is the address of a buffer of n characters to hold the resulting string. High and low contain the most and least significant 32 bits of the floating point value, respectively. ieleedftos returns the number of characters in the resulting string.

SOURCE
/sys/src/libmach

SEE ALSO
mach(2), symbol(2), errstr(2)

DIAGNOSTICS
Set errstr.
NAME
setupDESstate, des_key_setup, block_cipher, desCBCencrypt, desCBCdecrypt, desECBencrypt,
desECBdecrypt, des3CBCencrypt, des3CBCdecrypt, des3ECBencrypt, des3ECBdecrypt, key_setup,
des56to64, des64to56, setupDES3state, triple_block_cipher – single and triple digital encryption
standard

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <mp.h>
#include <libsec.h>

void des_key_setup(uchar key[8], ulong schedule[32])
void block_cipher(ulong *schedule, uchar *data, int decrypting)
void setupDESstate(DESstate *s, uchar key[8], uchar *ivec)
void desCBCencrypt(uchar *p, int len, DESstate *s)
void desCBCdecrypt(uchar *p, int len, DESstate *s)
void desECBencrypt(uchar *p, int len, DESstate *s)
void desECBdecrypt(uchar *p, int len, DESstate *s)
void triple_block_cipher(ulong expanded_key[3][32], uchar text[8],
int ende)
void setupDES3state(DES3state *s, uchar key[3][8], uchar *ivec)
void des3CBCencrypt(uchar *p, int len, DES3state *s)
void des3CBCdecrypt(uchar *p, int len, DES3state *s)
void des3ECBencrypt(uchar *p, int len, DES3state *s)
void des3ECBdecrypt(uchar *p, int len, DES3state *s)
void key_setup(uchar[7], ulong[32])
void des56to64(uchar *k64, uchar *k56)
void des64to56(uchar *k64, uchar *k56)

DESCRIPTION
The Digital Encryption Standard (DES) is a shared–key or symmetric encryption algorithm using
either a 56–bit key for single DES or three 56–bit keys for triple DES. The keys are encoded into
64 bits where every eight bit is parity.

The basic DES function, block_cipher, works on a block of 8 bytes, converting them in place. It
takes a key schedule, a pointer to the block, and a flag indicating encrypting (0) or decrypting (1).
The key schedule is created from the key using des_key_setup.

Since it is a bit awkward, block_cipher is rarely called directly. Instead, one normally uses routines
that encrypt larger buffers of data and which may chain the encryption state from one buffer to the
next. These routines keep track of the state of the encryption using a DESstate structure that
contains the key schedule and any chained state. SetupDESstate sets up the DESstate structure
using the key and an 8–byte initialization vector.

Electronic code book, using desECBencrypt and desECBdecrypt, is the less secure mode. The
encryption of each 8 bytes does not depend on the encryption of any other. Hence the encryption
is a substitution cipher using 64 bit characters.

Cipher block chaining mode, using desCBCencrypt and desCBCdecrypt, is more secure. Every
block encrypted depends on the initialization vector and all blocks encrypted before it.

For both CBC and ECB modes, a stream of data can be encrypted as multiple buffers. However, all
buffers except the last must be a multiple of 8 bytes to ensure successful decryption of the stream.
There are equivalent triple–DES (DES3–EDE) functions for each of the DES functions.
In the past, Plan 9 used a 56-bit or 7-byte format for DES keys. To be compatible with the rest of the world, we've abandoned this format. There are two functions, \texttt{des65to64} and \texttt{des64to56}, to convert back and forth between the two formats. Also a key schedule can be set up from the 7-byte format using \texttt{key_setup}.

\textbf{SOURCE}

/sys/src/libsec

\textbf{SEE ALSO}

\texttt{mp(2), aes(2), blowfish(2), dsa(2), elgamal(2), rc4(2), rsa(2), sechash(2), prime(2), rand(2)}

\textit{Breaking DES}, Electronic Frontier Foundation, O'Reilly, 1998

\textbf{BUGS}

Single DES can be realistically broken by brute-force; its 56-bit key is just too short. It should not be used in new code, which should probably use \texttt{aes(2)} instead, or at least triple DES.
NAME
dial, hangup, announce, listen, accept, reject, netmkaddr, setnetmpt, getnetconninfo, freenetconninfo – make and break network connections

SYNOPSIS
#include <u.h>
#include <libc.h>
int dial(char *addr, char *local, char *dir, int *cfdp)
int hangup(int ctl)
int announce(char *addr, char *dir)
int listen(char *dir, char *newdir)
int accept(int ctl, char *dir)
int reject(int ctl, char *dir, char *cause)
char* netmkaddr(char *addr, char *defnet, char *defservice)
void setnetmpt(char *to, int tolen, char *from)
NetConnInfo* getnetconninfo(char *conndir, int fd)
void freenetconninfo(NetConnInfo*)

DESCRIPTION
For these routines, *addr is a network address of the form network!netaddr!service, network!netaddr, or simply netaddr. Network is any directory listed in /net or the special token, net. Net is a free variable that stands for any network in common between the source and the host netaddr. Netaddr can be a host name, a domain name, a network address, or a meta-name of the form $attribute, which is replaced by value from the value-attribute pair attribute=value most closely associated with the source host in the network data base (see ndb(6)).

If a connection attempt is successful and *dir is non-zero, the path name of a line directory that has files for accessing the connection is copied into *dir. The path name is guaranteed to be less than 40 bytes long. One line directory exists for each possible connection. The data file in the line directory should be used to communicate with the destination. The ctl file in the line directory can be used to send commands to the line. See ip(3) for messages that can be written to the ctl file. The last close of the data or ctl file will close the connection.

Dial makes a call to destination *addr on a multiplexed network. If the network in *addr is net, dial will try in succession all networks in common between source and destination until a call succeeds. It returns a file descriptor open for reading and writing the data file in the line directory. The addr file in the line directory contains the address called. If the network allows the local address to be set, as is the case with UDP and TCP port numbers, and local is non-zero, the local address will be set to local. If *cfdp is non-zero, *cfdp is set to a file descriptor open for reading and writing the control file.

Hangup is a means of forcing a connection to hang up without closing the ctl and data files.

Announce and listen are the complements of dial. Announce establishes a network name to which calls can be made. Like dial, announce returns an open ctl file. The netaddr used in announce may be a local address or an asterisk, to indicate all local addresses, e.g. tcp!*echo. The listen routine takes as its first argument the dir of a previous announce. When a call is received, listen returns an open ctl file for the line the call was received on. It sets newdir to the path name of the new line directory. Accept accepts a call received by listen, while reject refuses the call because of cause. Accept returns a file descriptor for the data file opened ORDWR.

Netmkaddr makes an address suitable for dialing or announcing. It takes an address along with a default network and service to use if they are not specified in the address. It returns a pointer to static data holding the actual address to use.

Getnetconninfo returns a structure containing information about a network connection. The structure is:
  typedef struct NetConnInfo NetConnInfo;
  struct NetConnInfo

345
### EXAMPLES

Make a call and return an open file descriptor to use for communications:

```c
int callkremvax(void)
{
    return dial("kremvax", 0, 0, 0);
}
```

Call the local authentication server:

```c
int dialauth(char *service)
{
    return dial(netmkaddr("$auth", 0, service), 0, 0, 0);
}
```

Announce as kremvax on TCP/IP and loop forever receiving calls and echoing back to the caller anything sent:

```c
int bekremvax(void)
{
    int dfd, acfd, lcfds;
    char adir[40], ldir[40];
    int n;
    char buf[256];

    acfd = announce("tcp!*!7", adir);
    if(acfd < 0)
        return -1;
    for(;;){
        /* listen for a call */
        lcfds = listen(adir, ldir);
        if(lcfds < 0)
            return -1;
        /* fork a process to echo */
        switch(fork()){
            case -1:
                perror("forking");
                close(lcfds);
                return -1;
            case 0:
                break;
        }
    }
}
```
break;
case 0:
    /* accept the call and open the data file */
    fd = accept(lcfd, ldir);
    if(fd < 0)
        return -1;

    /* echo until EOF */
    while((n = read(fd, buf, sizeof(buf))) > 0)
        write(fd, buf, n);
    exits(0);
default:
    close(lcfd);
    break;
}

SOURCE
/sys/src/libc/9sys, /sys/src/libc/port

SEE ALSO
    auth(2), ip(3), ndb(8)

DIAGNOSTICS
    Dial, announce, and listen return -1 if they fail. Hangup returns nonzero if it fails.
NAME
dirread, dirreadall - read directory

SYNOPSIS
#include <u.h>
#include <libc.h>
long dirread(int fd, Dir **buf)
long dirreadall(int fd, Dir **buf)
#define STATMAX 65535U
#define DIRMAX (sizeof(Dir)+STATMAX)

DESCRIPTION
The data returned by a read(2) on a directory is a set of complete directory entries in a machine-
independent format, exactly equivalent to the result of a stat(2) on each file or subdirectory in the
directory. Dirread decodes the directory entries into a machine–dependent form. It reads from fd
and unpacks the data into an array of Dir structures whose address is returned in *buf (see
stat(2) for the layout of a Dir). The array is allocated with malloc(2) each time dirread is called.
Dirreadall is like dirread, but reads in the entire directory; by contrast, dirread steps through a
directory one read(2) at a time.
Directory entries have variable length. A successful read of a directory always returns an integral
number of complete directory entries; dirread always returns complete Dir structures. See
read(5) for more information.
The constant STATMAX is the maximum size that a directory entry can occupy. The constant
DIRMAX is an upper limit on the size necessary to hold a Dir structure and all the associated
data.
Dirread and dirreadall return the number of Dir structures filled in buf. The file offset is
advanced by the number of bytes actually read.

SOURCE
/sys/src/libc/9sys/dirread.c

SEE ALSO
intro(2), open(2), read(2)

DIAGNOSTICS
Dirread and Dirreadall return zero for end of file and a negative value for error. In either case,
*buf is set to nil so the pointer can always be freed with impunity.
These functions set errstr.
NAME
opendisk, Disk - generic disk device interface

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <disk.h>

typedef struct Disk {
    char *prefix;
    char part[NAMELEN];
    int fd, wfd, ctlfd, rdonly;
    int type;
    vlong secs, secsize, size, offset;
    int c, h, s;
} Disk;

Disk* opendisk(char *file, int rdonly, int noctl)

DESCRIPTION
These routines provide a simple way to gather and use information about floppy(3) and sd(3) disks and disk partitions, as well as plain files.

Opendedisk opens file for reading and stores the file descriptor in the fd field of the Disk structure. If rdonly is not set, opendisk also opens file for writing and stores that file descriptor in wfd. The two file descriptors are kept separate to help prevent accidents.

If noctl is not set, opendisk looks for a ctl file in the same directory as the disk file; if it finds one, it declares the disk to be an sd device, setting the type field in the Disk structure to Tsd. If the passed file is named fdnidisk, it looks for a file fdnctl, and if it finds that, declares the disk to be a floppy disk, of type Tfloppy. If either control file is found, it is opened for reading and writing, and the resulting file descriptor is saved as ctlfd. Otherwise the returned disk has type Tfile.

Opendedisk then stats the file and stores its length in size. If the disk is an sd partition, opendisk reads the sector size from the control file and stores it in secsize; otherwise the sector size is assumed to be 512, as is the case for floppy disks. Opendedisk then stores the disk size measured in sectors in secs.

If the disk is an sd partition, opendisk parses the control file to find the partition’s offset within its disk; otherwise it sets offset to zero. If the disk is an ATA disk, opendisk reads the disk geometry (number of cylinders, heads, and sectors) from the geometry line in the sd control file; otherwise it sets these to zero as well. Name is initialized with the base name of the disk partition, and is useful for forming messages to the sd control file. Prefix is set to the passed filename without the name suffix.

The IBM PC BIOS interface allocates 10 bits for the number of cylinders, 8 for the number of heads, and 6 for the number of sectors per track. Disk geometries are not quite so simple anymore, but to keep the interface useful, modern disks and BIOSes present geometries that still fit within these constraints. These numbers are still used when partitioning and formatting disks. Opendedisk employs a number of heuristics to discover this supposed geometry and store it in the c, h, and s fields. Disk offsets in partition tables and in FAT descriptors are stored in a form dependent upon these numbers, so opendisk works hard to report numbers that agree with those used by other operating systems; the numbers bear little or no resemblance to reality.

SOURCE
/sys/src/libdisk/disk.c

SEE ALSO
floppy(3), sd(3)
NAME
Image, draw, gendraw, drawreplxy, drawrepl, replclipr, line, poly, fillpoly, bezier, bezspline, fill-
bezier, fillbezspline, ellipse, fillellipse, arc, fillarc, icossin, icossin2, border, string, stringn, rune-
string, runestringn, stringbg, stringnbg, runestringbg, runestringnbg, _string, ARROW, drawsetde-
bug – graphics functions

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
typedef
struct Image
{
    Display *display; /* display holding data */
    int id; /* id of system−held Image */
    Rectangle r; /* rectangle in data area, local coords */
    Rectangle clipr; /* clipping region */
    ulong chan; /* pixel channel format descriptor */
    int depth; /* number of bits per pixel */
    int repl; /* flag: data replicates to tile clipr */
    Screen *screen; /* 0 if not a window */
    Image *next; /* next in list of windows */
} Image;
typedef enum
{
    /* Porter−Duff compositing operators */
    Clear = 0,
    SinD = 8,
    DinS = 4,
    SoutD = 2,
    DoutS = 1,
    S = SinD|SoutD,
    SoverD = SinD|SoutD|DoutS,
    SatopD = SinD|DoutS,
    SxorD = SoutD|DoutS,
    D = DinS|DoutS,
    DoverS = DinS|DoutS|SoutD,
    DatopS = DinS|SoutD,
    DxorS = DoutS|SoutD, /* == SxorD */
    Ncomp = 12,
} Drawop;

void draw(Image *dst, Rectangle r, Image *src,
    Image *mask, Point p)
void drawop(Image *dst, Rectangle r, Image *src,
    Image *mask, Point p, Drawop op)
void gendraw(Image *dst, Rectangle r, Image *src, Point sp,
    Image *mask, Point mp)
void gendrawop(Image *dst, Rectangle r, Image *src, Point sp,
    Image *mask, Point mp, Drawop op)
int drawreplxy(int min, int max, int x)
Point drawrepl(Rectangle r, Point p)
void replclipr(Image *i, int repl, Rectangle clipr)
void line(Image *dst, Point p0, Point p1, int end0, int end1,
    int radius, Image *src, Point sp)
void lineop(Image *dst, Point p0, Point p1, int end0, int end1,
    int radius, Image *src, Point sp, Drawop op)
void poly(Image *dst, Point *p, int np, int end0, int end1, int radius, Image *src, Point sp)
void polyop(Image *dst, Point *p, int np, int end0, int end1, int radius, Image *src, Point sp, Drawop op)
void fillpoly(Image *dst, Point *p, int np, int wind, Image *src, Point sp)
void fillpolyop(Image *dst, Point *p, int np, int wind, Image *src, Point sp, Drawop op)
int bezier(Image *dst, Point p0, Point p1, Point p2, Point p3, int end0, int end1, int radius, Image *src, Point sp)
int bezierop(Image *dst, Point p0, Point p1, Point p2, Point p3, int end0, int end1, int radius, Image *src, Point sp, Drawop op)
int bezspline(Image *dst, Point *pt, int npt, int end0, int end1, int radius, Image *src, Point sp)
int bezsplineop(Image *dst, Point *pt, int npt, int end0, int end1, int radius, Image *src, Point sp, Drawop op)
int bezsplinepts(Point *pt, int npt, Point **pp)
infillbezier(Image *dst, Point p0, Point p1, Point p2, Point p3, int w, Image *src, Point sp)
infillbezierop(Image *dst, Point p0, Point p1, Point p2, Point p3, int w, Image *src, Point sp, Drawop op)
infillbezspline(Image *dst, Point *pt, int npt, int w, Image *src, Point sp)
infillbezsplineop(Image *dst, Point *pt, int npt, int w, Image *src, Point sp, Drawop op)
void ellipse(Image *dst, Point c, int a, int b, int thick, Image *src, Point sp, Drawop op)
infillellipse(Image *dst, Point c, int a, int b, Image *src, Point sp)
infillellipseop(Image *dst, Point c, int a, int b, Image *src, Point sp, Drawop op)
infillarc(Image *dst, Point c, int a, int b, Image *src, Point sp, int alpha, int phi)
infillarcop(Image *dst, Point c, int a, int b, Image *src, Point sp, int alpha, int phi, Drawop op)
int icossin(int deg, int *cosp, int *sinp)
inicossin2(int x, int y, int *cosp, int *sinp)
void border(Image *dst, Rectangle r, int i, Image *color, Point sp)
Point string(Image *dst, Point p, Image *src, Point sp, Font *f, char *s)
Point stringop(Image *dst, Point p, Image *src, Point sp, Font *f, char *s, Drawop op)
Point stringn(Image *dst, Point p, Image *src, Point sp, Font *f, char *s)
Point stringnop(Image *dst, Point p, Image *src, Point sp, Font *f, char *s, int len)
Point runestring(Image *dst, Point p, Image *src, Point sp, Font *f, Rune *r)
Point runestringop(Image *dst, Point p, Image *src, Point sp, Font *f, Rune *r, Drawop op)
Point runestringn(Image *dst, Point p, Image *src, Point sp, Font *f, Rune *r, int len)
Point runestringnop(Image *dst, Point p, Image *src, Point sp, Font *f, Rune *r, int len, Drawop op)
Point stringbg(Image *dst, Point p, Image *src, Point sp, Font *f, Rune *r, Image *bg, Point bgp)
Point stringbgop(Image *dst, Point p, Image *src, Point sp, Font *f, char *s, Image *bg, Point bgp, Drawop op)
Point stringnb(Image *dst, Point p, Image *src, Point sp, Font *f, char *s, int len, Image *bg, Point bgp)
Point stringnbop(Image *dst, Point p, Image *src, Point sp, Font *f, char *s, int len, Image *bg, Point bgp, Drawop op)
Point runestringbg(Image *dst, Point p, Image *src, Point sp, Font *f, Rune *r, Image *bg, Point bgp)
Point runestringbgop(Image *dst, Point p, Image *src, Point sp, Font *f, Rune *r, Image *bg, Point bgp, Drawop op)
Point runestringnbg(Image *dst, Point p, Image *src, Point sp, Font *f, Rune *r, int len, Image *bg, Point bgp)
Point runestringnbgop(Image *dst, Point p, Image *src, Point sp, Font *f, Rune *r, int len, Image *bg, Point bgp, Drawop op)
Point _string(Image *dst, Point p, Image *src, Point sp, Font *f, char *s, Rune *r, int len, Rect clipr, Image *bg, Point bgp, Drawop op)

void drawsetdebug(int on)
enum
{
   /* line ends */
   Endsquare = 0,
   Enddisc = 1,
   Endarrow= 2,
   Endmask = 0x1F
};
#define ARROW(a, b, c) (Endarrow|((a)<<5)|((b)<<14)|((c)<<23))

DESCRIPTION
The Image type defines rectangular pictures and the methods to draw upon them; it is also the building block for higher level objects such as windows and fonts. In particular, a window is represented as an Image; no special operators are needed to draw on a window.

r   The coordinates of the rectangle in the plane for which the Image has defined pixel values. It should not be modified after the image is created.

clipr   The clipping rectangle: operations that read or write the image will not access pixels outside clipr. Frequently, clipr is the same as r, but it may differ; see in particular the discussion of repl. The clipping region may be modified dynamically using replclipr (q.v.).

chan   The pixel channel format descriptor, as described in image(6). The value should not be modified after the image is created.

depth   The number of bits per pixel in the picture; it is identically chantodepth(chan) (see graphics(2)) and is provided as a convenience. The value should not be modified after the image is created.

repl   A boolean value specifying whether the image is tiled to cover the plane when used as a source for a drawing operation. If repl is zero, operations are restricted to the intersection of r and clipr. If repl is set, r defines the tile to be replicated and clipr defines the portion of the plane covered by the tiling, in other words, r is replicated to cover clipr; in such cases r and clipr are independent.

For example, a replicated image with r set to ((0, 0), (1, 1)) and clipr set to ((0, 0), (1000, 1000)), with the single pixel of r set to blue, behaves identically to an image with r and clipr both set to ((0, 0), (1000, 1000)) and all pixels set to blue. However, the first image requires far less memory. The replication flag may be modified dynamically using replclipr (q.v.).
Most of the drawing functions come in two forms: a basic form, and an extended form that takes an extra Drawop to specify a Porter–Duff compositing operator to use. The basic forms assume the operator is Over1D, which suffices for the vast majority of applications. The extended forms are named by adding an –op suffix to the basic form. Only the basic forms are listed below.

\[\text{draw}(\text{dst, r, src, mask, p})\]

Draw is the standard drawing function. Only those pixels within the intersection of \(\text{dst} \rightarrow \text{r}\) and \(\text{dst} \rightarrow \text{clipr}\) will be affected; \text{draw} ignores \(\text{dst} \rightarrow \text{repl}\). The operation proceeds as follows (this is a description of the behavior, not the implementation):

1. If \(\text{repl}\) is set in \(\text{src}\) or \(\text{mask}\), replicate their contents to fill their clip rectangles.
2. Translate \(\text{src}\) and \(\text{mask}\) so \(p\) is aligned with \(r \cdot \text{min}\).
3. Set \(r\) to the intersection of \(r\) and \(\text{dst} \rightarrow \text{r}\).
4. Intersect \(r\) with \(\text{src} \rightarrow \text{clipr}\). If \(\text{src} \rightarrow \text{repl}\) is false, also intersect \(r\) with \(\text{src} \rightarrow \text{r}\).
5. Intersect \(r\) with \(\text{mask} \rightarrow \text{clipr}\). If \(\text{mask} \rightarrow \text{repl}\) is false, also intersect \(r\) with \(\text{mask} \rightarrow \text{r}\).
6. For each location in \(r\), combine the \(\text{dst}\) pixel with the \(\text{src}\) pixel using the alpha value corresponding to the \(\text{mask}\) pixel. If the \(\text{mask}\) has an explicit alpha channel, the alpha value corresponding to the \(\text{mask}\) pixel is simply that pixel’s alpha channel. Otherwise, the alpha value is the NTSC grayscale equivalent of the color value, with white meaning opaque and black transparent. In terms of the Porter–Duff compositing algebra, \text{draw} replaces the \(\text{dst}\) pixels with \((\text{src} \in \text{mask}) \over \text{dst}\). (In the extended form, “over” is replaced by \text{op}).

The various pixel channel formats involved need not be identical. If the channels involved are smaller than 8–bits, they will be promoted before the calculation by replicating the extant bits; after the calculation, they will be truncated to their proper sizes.

\[\text{gendraw}(\text{dst, r, src, p0, mask, p1})\]

Similar to \text{draw} except that \text{gendraw} aligns the source and mask differently: \(\text{src}\) is aligned so \(p0\) corresponds to \(r \cdot \text{min}\) and \(\text{mask}\) is aligned so \(p1\) corresponds to \(r \cdot \text{min}\). For most purposes with simple masks and source images, \text{draw} is sufficient, but \text{gendraw} is the general operator and the one all other drawing primitives are built upon.

\[\text{drawreplxy}(\text{min, max, x})\]

Clips \(x\) to be in the half-open interval \([\text{min}, \text{max})\) by adding or subtracting a multiple of \(\text{max} - \text{min}\).

\[\text{drawrepl}(\text{r, p})\]

Clips the point \(p\) to be within the rectangle \(r\) by translating the point horizontally by an integer multiple of rectangle width and vertically by the height.

\[\text{replclipr}(i, \text{repl}, \text{clipr})\]

Because the image data is stored on the server, local modifications to the Image data structure itself will have no effect. \text{Replclipr} modifies the local Image data structure’s repl and clipr fields, and notifies the server of their modification.

\[\text{line}(\text{dst, p0, p1, end0, end1, thick, src, sp})\]

Line draws in \(\text{dst}\) a line of width \(1 + 2 \cdot \text{thick}\) pixels joining points \(p0\) and \(p1\). The line is drawn using pixels from the src image aligned so \(sp\) in the source corresponds to \(p0\) in the destination. The line touches both \(p0\) and \(p1\), and \(\text{end0}\) and \(\text{end1}\) specify how the ends of the line are drawn. \text{Endsquare} terminates the line perpendicularly to the direction of the line; a thick line with \text{Endsquare} on both ends will be a rectangle. \text{Enddisc} terminates the line by drawing a disc of diameter \(1 + 2 \cdot \text{thick}\) centered on the end point. \text{Endarrow} terminates the line with an arrowhead whose tip touches the endpoint.

The macro \text{ARROW} permits explicit control of the shape of the arrow. If all three parameters are zero, it produces the default arrowhead, otherwise, \(a\) sets the distance along line from end of the regular line to tip, \(b\) sets the distance along line from the barb to the tip, and \(c\) sets the distance perpendicular to the line from edge of line to the tip of the barb, all in pixels.
Line and the other geometrical operators are equivalent to calls to \texttt{gendraw} using a mask produced by the geometric procedure.

\texttt{poly}(\texttt{dst, p, np, end0, end1, thick, src, sp})
Poly draws a general polygon; it is conceptually equivalent to a series of calls to \texttt{line} joining adjacent points in the array of Points \texttt{p}, which has \texttt{np} elements. The ends of the polygon are specified as in \texttt{line}; interior lines are terminated with \texttt{EndDimSc} to make smooth joins. The source is aligned so \texttt{sp} corresponds to \texttt{p[0]}.

\texttt{fillpoly}(\texttt{dst, p, np, wind, src, sp})
\texttt{Fillpoly} is like \texttt{poly} but fills in the resulting polygon rather than outlining it. The source is aligned so \texttt{sp} corresponds to \texttt{p[0]}. The winding rule parameter \texttt{wind} resolves ambiguities about what to fill if the polygon is self-intersecting. If \texttt{wind} is \texttt{~0}, a pixel is inside the polygon if the polygon's winding number about the point is non-zero. If \texttt{wind} is \texttt{1}, a pixel is inside if the winding number is odd. Complementary values (0 or \texttt{~1}) cause outside pixels to be filled. The meaning of other values is undefined. The polygon is closed with a line if necessary.

\texttt{bezier}(\texttt{dst, a, b, c, d, end0, end1, thick, src, sp})
\texttt{Bezier} draws the cubic Bezier curve defined by Points \texttt{a}, \texttt{b}, \texttt{c}, and \texttt{d}. The end styles are determined by \texttt{end0} and \texttt{end1}; the thickness of the curve is 1+2*\texttt{thick}. The source is aligned so \texttt{sp} in \texttt{src} corresponds to \texttt{a} in \texttt{dst}.

\texttt{bez spline}(\texttt{dst, p, end0, end1, thick, src, sp})
\texttt{Bezspline} takes the same arguments as \texttt{poly} but draws a quadratic B-spline (despite its name) rather than a polygon. If the first and last points in \texttt{p} are equal, the spline has periodic end conditions.

\texttt{bez spline pts}(\texttt{pt, npt, pp})
\texttt{Bez spline pts} returns in \texttt{pp} a list of points making up the open polygon that \texttt{bez spline} would draw. The caller is responsible for freeing \texttt{pp}.

\texttt{fillbezier}(\texttt{dst, a, b, c, d, wind, src, sp})
\texttt{Fillbezier} is to \texttt{bezier} as \texttt{fillpoly} is to \texttt{poly}.

\texttt{fillbez spline}(\texttt{dst, p, wind, src, sp})
\texttt{Fillbezspline} is like \texttt{fillpoly} but fills the quadratic B-spline rather than the polygon outlined by \texttt{p}. The spline is closed with a line if necessary.

\texttt{ellipse}(\texttt{dst, c, a, b, thick, src, sp})
\texttt{Ellipse} draws in \texttt{dst} an ellipse centered on \texttt{c} with horizontal and vertical semiaxes \texttt{a} and \texttt{b}. The source is aligned so \texttt{sp} in \texttt{src} corresponds to \texttt{c} in \texttt{dst}. The ellipse is drawn with thickness 1+2*\texttt{thick}.

\texttt{fill ellipse}(\texttt{dst, c, a, b, src, sp})
\texttt{Fillellipse} is like \texttt{ellipse} but fills the ellipse rather than outlining it.

\texttt{arc}(\texttt{dst, c, a, b, thick, src, sp, alpha, phi})
\texttt{Arc} is like \texttt{ellipse}, but draws only that portion of the ellipse starting at angle \texttt{alpha} and extending through an angle of \texttt{phi}. The angles are measured in degrees counterclockwise from the positive x axis.

\texttt{fill arc}(\texttt{dst, c, a, b, src, sp, alpha, phi})
\texttt{Fillarc} is like \texttt{arc}, but fills the sector with the source color.

\texttt{icossin}(\texttt{deg, cosp, sinp})
\texttt{Icossin} stores in \texttt{*cosp} and \texttt{*sinp} scaled integers representing the cosine and sine of the angle \texttt{deg}, measured in integer degrees. The values are scaled so \texttt{cos(0)} is 1024.

\texttt{icossin2}(\texttt{x, y, cosp, sinp})
\texttt{Icossin2} is analogous to \texttt{icossin}, with the angle represented not in degrees but implicitly by the point \texttt{(x,y)}. It is to \texttt{icossin} what \texttt{atan2} is to \texttt{atan} (see \texttt{sin(2)}).

\texttt{border}(\texttt{dst, r, i, color, sp})
\texttt{Border} draws an outline of rectangle \texttt{r} in the specified \texttt{color}. The outline has width \texttt{i}; if positive, the border goes inside the rectangle; negative, outside. The source is aligned so \texttt{sp} corresponds to \texttt{r.min}.
string(dst, p, src, sp, font, s)

String draws in dst characters specified by the string s and font; it is equivalent to a series of calls to gendraw using source src and masks determined by the character shapes. The text is positioned with the left of the first character at $p.x$ and the top of the line of text at $p.y$. The source is positioned so sp in src corresponds to p in dst. String returns a Point that is the position of the next character that would be drawn if the string were longer.

For characters with undefined or zero–width images in the font, the character at font position 0 (NUL) is drawn.

The other string routines are variants of this basic form, and have names that encode their variant behavior. Routines whose names contain Rune accept a string of Runes rather than UTF–encoded bytes. Routines ending in n accept an argument, n, that defines the number of characters to draw rather than accepting a NUL–terminated string. Routines containing bg draw the background behind the characters in the specified color (bg) and alignment (bgp); normally the text is drawn leaving the background intact.

The routine _string captures all this behavior into a single operator. Whether it draws a UTF string or Rune string depends on whether s or r is null (the string length is always determined by len). If bg is non–null, it is used as a background color. The clipr argument allows further management of clipping when drawing the string; it is intersected with the usual clipping rectangles to further limit the extent of the text.

drawsetdebug(on)

Turns on or off debugging output (usually to a serial line) according to whether on is non-zero.

SOURCE
/sys/src/libdraw

SEE ALSO
graphics(2), stringsize(2), color(6), utf(6), addpt(2)


DIAGNOSTICS
These routines call the graphics error function on fatal errors.

BUGS
Anti–aliased characters can be drawn by defining a font with multiple bits per pixel, but there are no anti–aliasing geometric primitives.
NAME
dsagen, dsasign, dsaverify, dsapuballoc, dsapubfree, dsaprivalloc, dsaprivfree, dsasigalloc, dsasigfree, dsaprivtopub – digital signature algorithm

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <mp.h>
#include <libsec.h>

DSApriv* dsagen(DSApub *opub)
DSAsig* dsasign(DSApriv *k, mpint *m)
int dsaverify(DSApub *k, DSAsig *sig, mpint *m)
DSApub* dsapuballoc(void)
void dsapubfree(DSApub*)
DSApriv* dsaprivalloc(void)
void dsaprivfree(DSApriv*)
DSAsig* dsasigalloc(void)
void dsasigfree(DSAsig*)
DSApub* dsaprivtopub(DSApriv*)

DESCRIPTION
DSA is the NIST approved digital signature algorithm. The owner of a key publishes the public part of the key:

```c
struct DSApub
{
    mpint   *p;   // modulus
    mpint   *q;   // group order, q divides p−1
    mpint   *alpha; // group generator
    mpint   *key;  // alpha**secret mod p
};
```

This part can be used for verifying signatures (with `dsaverify`) created by the owner. The owner signs (with `dsasign`) using his private key:

```c
struct DSApriv
{
    DSApub   pub;
    mpint   *secret; // (decryption key)
};
```

Keys are generated using `dsagen`. If `dsagen`'s argument `opub` is nil, a key is created using a new p and q generated by `DSAprimes` (see `prime(2)`). Otherwise, p and q are copied from the old key.

`Dsaprivtopub` returns a newly allocated copy of the public key corresponding to the private key.

The routines `dsapuballoc`, `dsapubfree`, `dsaprivalloc`, and `dsaprivfree` are provided to manage key storage.

`Dsasign` signs message `m` using a private key `k` yielding a

```c
struct DSAsig
{
    mpint   *r, *s;
};
```

`Dsaverify` returns 0 if the signature is valid and −1 if not.

The routines `dsasigalloc` and `dsasigfree` are provided to manage signature storage.

SOURCE
/sys/src/libsec
SEE ALSO

\textit{mp(2)}, \textit{aes(2)}, \textit{blowfish(2)}, \textit{des(2)}, \textit{rc4(2)}, \textit{rsa(2)}, \textit{sechash(2)}, \textit{prime(2)}, \textit{rand(2)}
NAME
dup - duplicate an open file descriptor

SYNOPSIS
#include <u.h>
#include <libc.h>
int dup(int oldfd, int newfd)

DESCRIPTION
Given a file descriptor, oldfd, referring to an open file, dup returns a new file descriptor referring
to the same file.

If newfd is -1 the system chooses the lowest available file descriptor. Otherwise, dup will use
newfd for the new file descriptor (closing any old file associated with newfd). File descriptors are
allocated dynamically, so to prevent unwarranted growth of the file descriptor table, dup requires
that newfd be no greater than 20 more than the highest file descriptor ever used by the program.

SOURCE
/sys/src/libc/9syscall

SEE ALSO
intro(2), dup(3)

DIAGNOSTICS
Sets errstr.
NAME

dynfindsym, dynfreeimport, dynloadfd, dynloadgen, dynobjfree, dyntabsize - load object file dynamically

SYNOPSIS

#include <u.h>
#include <libc.h>
#include <a.out.h>
#include <dynld.h>

Dynsym* dynfindsym(char *name, Dynsym *syms, int nsym);
Dynobj* dynloadfd(int fd, Dynsym *exports, int nexport, ulong maxsize);
Dynobj* dynloadgen(void *file, long (*read)(void*,void*,long),
                  vlong (*seek)(void*,vlong,int), void (*err)(char*),
                  Dynsym *exports, int nexport, ulong maxsize);
void* dynimport(Dynobj *o, char *name, ulong sig);
void dynfreeimport(Dynobj *o);
void dynobjfree(Dynobj *o);
int dyntabsize(Dynsym *t);

extern Dynsym __exporttab[];

DESCRIPTION

These functions allow a process to load further code and data into the currently executing image. A dynamically-loadable file, called a module here, is a variant of the a.out(6) executable format with some extra components. The loader for the architecture (see 2(l)) creates a module file from component object file(s) when given the -u option. A module contains text and data sections, an import table, an export table, and relocation data. The import table lists the symbols the module needs from the loading program; the export table lists symbols the module provides when loaded. A program that loads a module provides a table of its own symbols to match the symbols in the module's import table.

A symbol entry in a symbol table names a global function or data item, and has an associated signature value representing the type of the corresponding function or data in the source code. The Dynsym structure defines a symbol:

    typedef struct {
        ulong sig;
        ulong addr;
        char* name;
    } Dynsym;

The structure is known to the loaders 2(l). Name is the linkage name of the function or data. Addr is its address, which is relative to the start of the module before loading, and an address in the current address space after loading. The signature sig is the value produced by the C compiler's signof operator applied to the type. Symbol tables must be sorted by name.

An executable that wishes to load modules will normally be linked using the -x option to the appropriate loader 2(l). The resulting executable contains an export table __exporttab that lists all the exported symbols of the program (by default, all external symbols). A nil name marks the end of the table. See 2(l) for details. The table can be given to the functions below to allow a loaded module to access those symbols.

A loaded module is described by a Dynobj structure:

    typedef struct {
        ulong size; /* total size in bytes */
        ulong text; /* bytes of text */
        ulong data; /* bytes of data */
        ulong bss; /* bytes of bss */
        uchar* base; /* start of text, data, bss */
    } Dynobj,
int nexport;
Dynsym* export; /* export table */
int nimport;
Dynsym** import; /* import table */
} Dynobj;

Several fields give sizes of the module’s components, as noted in comments above. Base gives the address at which the module has been loaded. All its internal references have been adjusted where needed to reflect its current address. Export points to a symbol table listing the symbols exported by the module; nexport gives the table’s length. Import points to a list of symbols imported by the module; note that each entry actually points to an entry in a symbol table provided by the program that loaded the module (see below). Nimport gives the import table’s length. If the import table is not required, call dynfreeimport on the module pointer to free it.

Dynfindsym looks up the entry for the given name in symbol table sym (of length nsym). It returns a pointer to the entry if found; nil otherwise. The symbol table must be sorted by name in ascending order.

Dyntabsize returns the length of symbol table t, defined to be the number of Dynsym values starting at t that have non-nil name fields. It is used to find the length of _exporttab.

Dynloadfd loads a module from the file open for reading on fd, and returns the resulting module pointer on success, or nil on error. If maxsize is non-zero the size of the dynamically-loaded module’s code and data is limited to maxsize bytes. Exports is an array of nexport symbols in the current program that can be imported by the current module. It uses read(2) and seek(2) to access fd, and calls werrstr (see errstr(2)) to set the error string if necessary.

Dynloadgen is a more general function that can load a module from an arbitrary source, not just an open file descriptor. (In particular, it can be called by the kernel using functions internal to the kernel instead of making system calls.) Exports, nexport and maxsize are just as for dynloadfd. File is a pointer to a structure defined by the caller that represents the file containing the module. It is passed to read and seek. Read is invoked as (*read)(file, buf, nbytes). Read should read nbytes of data from file into buf and return the number of bytes transferred. It should return -1 on error. Seek is invoked as (*seek)(file, n, type) where n and type are just as for seek(2); it should seek to the requested offset in file, or return -1 on error. Dynloadgen returns a pointer to the loaded module on success. On error, it returns nil after calling its err parameter to set the error string.

Dynimport returns a pointer to the value of the symbol name in loaded module o, or nil if o does not export a symbol with the given name. If sig is non-zero, the exported symbol’s signature must equal sig, or dynimport again returns nil. For example:

```c
    Dev *d;
    d = dynimport(obj, "XXXdevtab", signof(*d));
    if(d == nil)
        error("not a dynamically-loadable driver");
```

Dynobjfree frees the module o. There is no reference counting; it is the caller’s responsibility to decide whether a module is no longer needed.

SEE ALSO
2l(1), mach(2), a.out(6)

DIAGNOSTICS
Functions that return pointers return nil on error. Dynloadfd sets the error string and returns nil.
NAME
eggen, egencrypt, egdecrypt, egsign, egverify, egpuballoc, egpubfree, egprivalloc, egprivfree, egsigalloc, egsigfree, egprivtopub – elgamal encryption

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <mp.h>
#include <libsec.h>
EGpriv* eggen(int nlen, int nrep)
mpint* egencrypt(EGpub *k, mpint *in, mpint *out)
mpint* egdecrypt(EGpriv *k, mpint *in, mpint *out)
EGsig* egsign(EGpriv *k, mpint *m)
int egverify(EGpub *k, EGsig *sig, mpint *m)
EGpub* egpuballoc(void)
void egpubfree(EGpub*)
EGpriv* egprivalloc(void)
void egprivfree(EGpriv*)
EGpub* egprivtopub(EGpriv*)

DESCRIPTION
Elgamal is a public key encryption and signature algorithm. The owner of a key publishes the public part of the key:

struct EGpub
{
    mpint *p; // modulus
    mpint *alpha; // generator
    mpint *key; // (encryption key) alpha**secret mod p
}

This part can be used for encrypting data (with egencrypt) to be sent to the owner. The owner decrypts (with egdecrypt) using his private key:

struct EGpriv
{
    EGpub pub;
    mpint *secret; // (decryption key)
}

Keys are generated using eggen. Eggen takes both bit length of the modulus and the number of repetitions of the Miller–Rabin primality test to run. If the latter is 0, it does the default number of rounds. Egprivtopub returns a newly allocated copy of the public key corresponding to the private key.

The routines egpuballoc, egpubfree, egprivalloc, and egprivfree are provided to manage key storage.

Egsign signs message m using a private key k yielding a

struct EGsig
{
    mpint *r, *s;
}

Egverify returns 0 if the signature is valid and -1 if not.

The routines egsigalloc and egsigfree are provided to manage signature storage.
SOURCE
/sys/src/libsec
SEE ALSO
mpi(2), aes(2), blowfish(2), des(2), dsa(2), rc4(2), rsa(2), sechash(2), prime(2), rand(2)
NAME
dec64, enc64, dec32, enc32, dec16, enc16, encodefmt - encoding byte arrays as strings

SYNOPSIS
#include <u.h>
#include <libc.h>

int dec64(uchar *out, int lim, char *in, int n)
int enc64(char *out, int lim, uchar *in, int n)
int dec32(uchar *out, int lim, char *in, int n)
int enc32(char *out, int lim, uchar *in, int n)
int dec16(uchar *out, int lim, char *in, int n)
int enc16(char *out, int lim, uchar *in, int n)
int encodefmt(Fmt*)

DESCRIPTION
Enc16, enc32 and enc64 create null terminated strings. They return the size of the encoded string (without the null) or -1 if the encoding fails. The encoding fails if \textit{lim}, the length of the output buffer, is too small.

Dec16, dec32 and dec64 return the number of bytes decoded or -1 if the decoding fails. The decoding fails if the output buffer is not large enough or, for base 32, if the input buffer length is not a multiple of 8.

Encodefmt can be used with \texttt{fmtinstall(2)} and \texttt{print(2)} to print encoded representations of byte arrays. The verbs are

- \texttt{H} base 16 (i.e. hexadecimal). The default encoding is in upper case. The \texttt{1} flag forces lower case.
- \texttt{<} base 32
- \texttt{[} base 64 (same as MIME)

The length of the array is specified as \texttt{f2}. For example, to display a 15 byte array as hex:

```c
char x[15];
fmtinstall('H', encodefmt);
print("%.H\n", sizeof x, x);
```

SOURCE
/sys/src/libc/port/u32.c
/sys/src/libc/port/u64.c
NAME
encrypt, decrypt, netcrypt – DES encryption

SYNOPSIS
#include <u.h>
#include <libc.h>
int encrypt(void *key, void *data, int len)
int decrypt(void *key, void *data, int len)
int netcrypt(void *key, void *data)

DESCRIPTION
Encrypt and decrypt perform DES encryption and decryption. Key is an array of DESKEYLEN (defined as 7 in <auth.h>) bytes containing the encryption key. Data is an array of len bytes; it must be at least 8 bytes long. The bytes are encrypted or decrypted in place.

The DES algorithm encrypts an individual 8-byte block of data. Encrypt uses the following method to encrypt data longer than 8 bytes. The first 8 bytes are encrypted as usual. The last byte of the encrypted result is prefixed to the next 7 unencrypted bytes to make the next 8 bytes to encrypt. This is repeated until fewer than 7 bytes remain unencrypted. Any remaining unencrypted bytes are encrypted with enough of the preceding encrypted bytes to make a full 8-byte block. Decrypt uses the inverse algorithm.

Netcrypt performs the same encryption as a SecureNet Key. Data points to an ASCII string of decimal digits with numeric value between 0 and 10000. These digits are copied into an 8-byte buffer with trailing binary zero fill and encrypted as one DES block. The first four bytes are each formatted as two digit ASCII hexadecimal numbers, and the string is copied into data.

SOURCE
/sys/src/libc/port

DIAGNOSTICS
These routines return 1 if the data was encrypted, and 0 if the encryption fails. Encrypt and decrypt fail if the data passed is less than 8 bytes long. Netcrypt can fail if it is passed invalid data.

SEE ALSO
securenet(8)

BUGS
The implementation is broken in a way that makes it unsuitable for anything but authentication.
NAME
waseror, poperror, nexterror, error, ferror, silenterror – exception handling for threaded programs

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <thread.h>
#include <error.h>

int waserror(void);
void poperror(void);
void nexterror(void);
void error(char *err);
void ferror(char *fmt, ...);
void silenterror(char *fmt, ...);

DESCRIPTION
The functions in this library provide an aexception handling mechanism modelled on that in the
Plan 9 kernel. A construct such as

try{
   ...
   if(...) raise(exception);
   ...
} except{
   handle exception
}

using this library becomes:
if(waserror()){
   handle exception
}
...
if(...) error("exception");
...
poperror();

Waserror and poperror are the bracketing elements around the code in which an exception might
be raised with a call to error.

Waserror sets a point to which control returns if an exception occurs and returns zero. If the
exception occurs, control transfers back to waserror and it then clears the point previously set and
returns non-zero.

Poperror clears the exception–return point previously set by waserror.

Error, ferror and silenterror all raise an error and they all set the error string. Ferror and
silenterror take a format string, while error just takes a simple string. Error and ferror print the
string on standard error as well. Silenterror does not.

Exception contexts bracketed by waserror and poperror can be nested. When an exception has
been handled in the innermost context, a call to nexterror transfers it to the next larger context.

EXAMPLES
Using exceptions to free dynamic memory:
if(waserror()){
   free(p);
   nexterror();
} 
p = malloc(something);
...
if(...)
   ferror("%s: %r", x);
...
free(p);
poperror();

Excerpt from the worker library. The worker calls a user-specified function that may raise an error. The error is caught and the worker prepares for the next customer:

```c
static void
worker(void *arg)
{
    Worker *w;

    w = arg;
    for(;;){
        w->r = recvp(w->chan);
        if(!waserror()){
            w->r->func(w, w->r->arg);
            poperror();
        }
        reqfree(w->r);
        sendp(workerthreads, w);
    }
}
```

**DIAGNOSTICS**

*Waserror* returns non-zero when an error was raised.

**SEE ALSO**

worker(2)

**BUGS**

The error stack is only 32 levels deep.

**AUTHOR**

Sape Mullender
NAME
errstr, rerrstr, werrstr - description of last system call error

SYNOPSIS
#include <u.h>
#include <libc.h>
int errstr(char *err, uint nerr)
void rerrstr(char *err, uint nerr)
void werrstr(char *fmt, ...)

DESCRIPTION
When a system call fails it returns -1 and records a null terminated string describing the error in a
per-process buffer. Errstr swaps the contents of that buffer with the contents of the array err.  Errstr will write at most nerr bytes into err; if the per-process error string does not fit, it is silently
truncated at a UTF character boundary. The returned string is NUL-terminated. Usually errstr will
be called with an empty string, but the exchange property provides a mechanism for libraries to
set the return value for the next call to errstr.

The per-process buffer is ERRMAX bytes long. Any error string provided by the user will be truncated at ERRMAX-1 bytes. ERRMAX is defined in <libc.h>.

If no system call has generated an error since the last call to errstr with an empty string, the result
is an empty string.

The verb \x in print(2) calls errstr and outputs the error string.

Rerrstr reads the error string but does not modify the per-process buffer, so a subsequent errstr
will recover the same string.

Werrstr takes a print style format as its argument and uses it to format a string to pass to errstr.
The string returned from errstr is discarded.

SOURCE
/sys/src/libc/9syscall
/sys/src/libc/9sys/werrstr.c

DIAGNOSTICS
Errstr always returns 0.

SEE ALSO
intro(2), perror(2)
NAME
  event, einit, estart, estartfn, etimer, eread, emouse, ekbd, ecanread, ecanmouse, ecankbd, eredmouse, eatomouse, eresized, egetrect, edrawgetrect, emenuhit, emoveto, esetcursor, Event, Mouse, Menu – graphics events

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  #include <draw.h>
  #include <event.h>
  #include <cursor.h>

  void  einit(ulong keys)
  ulong event(Event *e)
  Mouse emouse(void)
  int  ekbd(void)
  int  ecanmouse(void)
  int  ecankbd(void)
  int  eredmouse(Mouse *m)
  int  eatomouse(Mouse *m, char *buf, int n)
  ulong estart(ulong key, int fd, int n)
  ulong estartfn(int id, ulong key, int fd, int n,
      int (*fn)(Event*, uchar*, int))
  ulong etimer(ulong key, int n)
  ulong eread(ulong keys, Event *e)
  int  ecanread(ulong keys)
  void  eresized(int new)
  Rectangle egetrect(int but, Mouse *m)
  void  edrawgetrect(Rectangle r, int up)
  int  emenuhit(int but, Mouse *m, Menu *menu)
  int  emoveto(Point p)
  int  esetcursor(Cursor *c)

  extern Mouse  *mouse
  enum{
    Emouse = 1,
    Ekeyboard = 2,
  };

DESCRIPTION
  These routines provide an interface to multiple sources of input for unthreaded programs. Threaded programs (see thread(2)) should instead use the threaded mouse and keyboard interface described in mouse(2) and keyboard(2).

  Einit must be called first. If the argument to einit has the Emouse and Ekeyboard bits set, the mouse and keyboard events will be enabled; in this case, initdraw (see graphics(2)) must have already been called. The user must provide a function called eresized to be called whenever the window in which the process is running has been resized; the argument new is a flag specifying whether the program must call getwindow (see graphics(2)) to re-establish a connection to its window. After resizing (and perhaps calling getwindow), the global variable screen will be updated to point to the new window’s Image structure.

  As characters are typed on the keyboard, they are read by the event mechanism and put in a queue. Ekbd returns the next rune from the queue, blocking until the queue is non-empty. The
characters are read in raw mode (see \textit{cons}(3)), so they are available as soon as a complete rune is
typed.

When the mouse moves or a mouse button is pressed or released, a new mouse event is queued by
the event mechanism. \textit{Emouse} returns the next mouse event from the queue, blocking until the
queue is non-empty. \textit{Emouse} returns a Mouse structure:

\begin{verbatim}
struct Mouse {
    int    buttons;
    Point xy;
    ulong msec;
};
\end{verbatim}

Buttons\&1 is set when the left mouse button is pressed, buttons\&2 when the middle button
is pressed, and buttons\&4 when the right button is pressed. The current mouse position
always returned in \textit{xy}. Msec is a time stamp in units of milliseconds.

\textit{Ecankbd} and \textit{ecanmouse} return non-zero when there are keyboard or mouse events available to be
read.

\textit{Ereadmouse} reads the next mouse event from the file descriptor connected to the mouse, converts
the textual data into a Mouse structure by calling \textit{eatomouse} with the buffer and count from the
read call, and returns the number of bytes read, or -1 for an error.

\textit{Estart} can be used to register additional file descriptors to scan for input. It takes as arguments
the file descriptor to register, the maximum length of an event message on that descriptor, and a
key to be used in accessing the event. The key must be a power of 2 and must not conflict with
any previous keys. If a zero key is given, a key will be allocated and returned. \textit{Estartfn} is similar to
\textit{estart}, but processes the data received by calling \textit{fn} before returning the event to the user. The
function \textit{fn} is called with the \textit{id} of the event; it should return \textit{id} if the event is to be passed to the
user, 0 if it is to be ignored. The variable Event\texttt{.v} can be used by \textit{fn} to attach an arbitrary data
item to the returned Event structure. \textit{Ekepsilon} and \textit{Emouse} are the keyboard and mouse
event keys.

\textit{Etimer} starts a repeating timer with a period of \textit{n} milliseconds; it returns the timer event key, or
zero if it fails. Only one timer can be started. Extra timer events are not queued and the timer
channel has no associated data.

\textit{Eread} waits for the next event specified by the mask \textit{keys} of event keys submitted to \textit{estart}. It fills
in the appropriate field of the argument Event\texttt{.v} structure, which looks like:

\begin{verbatim}
struct Event {
    int    kbdc;
    Mouse mouse;
    int    n;
    void   *v;
    uchar  data[EMAXMSG];
};
\end{verbatim}

Data is an array which is large enough to hold a 9P message. \textit{Eread} returns the key for the event
which was chosen. For example, if a mouse event was read, Emouse will be returned.

\textit{Event} waits for the next event of any kind. The return is the same as for \textit{eread}.

As described in \textit{graphics}(2), the graphics functions are buffered. Event, eread, emouse, and ekbd
all cause a buffer flush unless there is an event of the appropriate type already queued.

\textit{Ecanread} checks whether a call to eread(\textit{keys}) would block, returning 0 if it would, 1 if it
would not.

\textit{Getrect} prompts the user to sweep a rectangle. It should be called with \textit{m} holding the mouse
event that triggered the egetrect (or, if none, a Mouse with buttons set to 7). It changes to the
sweep cursor, waits for the buttons all to be released, and then waits for button number \textit{but} to be
pressed, marking the initial corner. If another button is pressed instead, egetrect returns a rectan-
gle with zero for both corners, after waiting for all the buttons to be released. Otherwise, egetrect
continually draws the swept rectangle until the button is released again, and returns the swept
rectangle. The mouse structure pointed to by m will contain the final mouse event.

`Egetrect` uses successive calls to `edrawgetrect` to maintain the red rectangle showing the sweep-in-progress. The rectangle to be drawn is specified by rc and the up parameter says whether to draw (1) or erase (0) the rectangle.

`Emenuhit` displays a menu and returns a selected menu item number. It should be called with m holding the mouse event that triggered the `emenuhit`; it will call `emouse` to update it. A Menu is a structure:

```c
struct Menu
{
    char **item;
    char *(*gen)(int);
    int lasthit;
};
```

If item is nonzero, it should be a null-terminated array of the character strings to be displayed as menu items. Otherwise, gen should be a function that, given an item number, returns the character string for that item, or zero if the number is past the end of the list. Items are numbered starting at zero. `Menuhit` waits until but is released, and then returns the number of the selection, or -1 for no selection. The m argument is filled in with the final mouse event.

`Emoveto` moves the mouse cursor to the position p on the screen.

`Esetcursor` changes the cursor image to that described by the Cursor c (see `mouse(2)`). If c is nil, it restores the image to the default arrow.

**SOURCE**

/sys/src/libdraw

**SEE ALSO**

`rio(1), graphics(2), plumb(2), cons(3), draw(3)`
NAME
exec, execl, _privates, _nprivates, _tos – execute a file

SYNOPSIS
#include <u.h>
#include <libc.h>
void* exec(char *name, char* argv[])
void* execl(char *name, ...)
void **_privates;
int _nprivates;
#include <tos.h>
typedef struct Tos Tos;
struct Tos {
  struct { ... } prof; /* profiling data */
  ulong cyclefreq; /* cycle clock frequency */
  long kcycles; /* kernel cycles */
  long pcycles; /* process cycles (kernel + user) */
  ulong pid; /* process id */
  ulong clock; /* profiling clock */
  /* top of stack is here */
};
extern Tos *_tos;

DESCRIPTION
Exec and execl overlay the calling process with the named file, then transfer to the entry point of the image of the file.

Name points to the name of the file to be executed; it must not be a directory, and the permissions must allow the current user to execute it (see stat(2)). It should also be a valid binary image, as defined in the a.out(6) for the current machine architecture, or a shell script (see rc(1)). The first line of a shell script must begin with #! followed by the name of the program to interpret the file and any initial arguments to that program, for example

#!/bin/rc
ls | mc

When a C program is executed, it is called as follows:

void main(int argc, char *argv[])

Argv is a copy of the array of argument pointers passed to exec; that array must end in a null pointer, and argc is the number of elements before the null pointer. By convention, the first argument should be the name of the program to be executed. Execl is like exec except that argv will be an array of the parameters that follow name in the call. The last argument to execl must be a null pointer.

For a file beginning #!, the arguments passed to the program (/bin/rc in the example above) will be the name of the file being executed, any arguments on the #! line, the name of the file again, and finally the second and subsequent arguments given to the original exec call. The result honors the two conventions of a program accepting as argument a file to be interpreted and argv[0] naming the file being executed.

Most attributes of the calling process are carried into the result; in particular, files remain open across exec (except those opened with OEXEC or’d into the open mode; see open(2)); and the working directory and environment (see env(3)) remain the same. However, a newly exec’ed process has no notification handler (see notify(2)).

The global cell _privates points to an array of _nprivates elements of per-process private data. This storage is private for each process, even if the processes share data segments.

When the new program begins, the global pointer _tos is set to the address of a structure that holds information allowing accurate time keeping and clock reading in user space. These data are
updated by the kernel during of the life of the process, including across rfork and execs. If there is a user-space accessible fast clock (a processor cycle counter), cyclefreq will be set to its frequency in Hz. Kcycles (pcycles) counts the number of cycles this process has spent in kernel mode (kernel and user mode). Pid is the current process’s id. Clock is the user-profiling clock (see prof(1)). Its time is measured in milliseconds but is updated at a system-dependent lower rate. This clock is typically used by the profiler but is available to all programs.

The above conventions apply to C programs; the raw system interface to the new image is as follows: the word pointed to by the stack pointer is argc; the words beyond that are the zeroth and subsequent elements of argv, followed by a terminating null pointer; and the return register (e.g. R0 on the 68020) contains the address of the clock information.

SOURCE
/sys/src/libc/9syscall
/sys/src/libc/port/execl.c

SEE ALSO
prof(1), intro(2), stat(2)

DIAGNOSTICS
If these functions fail, they return and set errstr. There can be no return to the calling process from a successful exec or execl; the calling image is lost.

BUGS
There is a large but finite limit on the size of an argument list, typically around 409,600 bytes. The kernel constant TSTKSIZ controls this.
NAME
exits, _exits, atexit, atexitdont, terminate – terminate process, process cleanup

SYNOPSIS
#include <u.h>
#include <libc.h>
void _exits(char *msg)
void exits(char *msg)
int atexit(void(*)(void))
void atexitdont(void(*)(void))

DESCRIPTION
Exits is the conventional way to terminate a process. _Exits is the underlying system call. They can never return.

_Msg conventionally includes a brief (maximum length ERRLEN) explanation of the reason for exiting, or a null pointer or empty string to indicate normal termination. The string is passed to the parent process, prefixed by the name and process id of the exiting process, when the parent does a wait(2).

Before calling _exits with _msg as an argument, exits calls in reverse order all the functions recorded by atexit.

Atexit records fn as a function to be called by exits. It returns zero if it failed, nonzero otherwise. A typical use is to register a cleanup routine for an I/O package. To simplify programs that fork or share memory, exits only calls those atexit–registered functions that were registered by the same process as that calling exits.

Calling atexit twice (or more) with the same function argument causes exits to invoke the function twice (or more).

There is a limit to the number of exit functions that will be recorded; atexit returns 0 if that limit has been reached.

Atexitdont cancels a previous registration of an exit function.

SOURCE
/sys/src/libc/port/atexit.c

SEE ALSO
fork(2), wait(2)
NAME
exp, log, log10, pow, pow10, sqrt - exponential, logarithm, power, square root

SYNOPSIS
#include <u.h>
#include <libc.h>
double exp(double x)
double log(double x)
double log10(double x)
double pow(double x, double y)
double pow10(int n)
double sqrt(double x)

DESCRIPTION
Exp returns the exponential function of x.
Log returns the natural logarithm of x; log10 returns the base 10 logarithm.
Pow returns \( x^y \) and pow10 returns \( 10^n \) as a double.
Sqrt returns the square root of x.

SOURCE
All these routines have portable C implementations in /sys/src/libc/port. Most also have
machine-dependent implementations, written either in assembler or C, in
/sys/src/libc/$objtype.

SEE ALSO
hypot(2), sinh(2), intro(2)
NAME
fauth – set up authentication on a file descriptor to a file server

SYNOPSIS
#include <u.h>
#include <libc.h>
int fauth(int fd, char *aname)

DESCRIPTION
Fauth is used to establish authentication for the current user to access the resources available
through the 9P connection represented by fd. The return value is a file descriptor, conventionally
called afd, that is subsequently used to negotiate the authentication protocol for the server, typi-
cally using auth_proxy or fauth_proxy (see auth(2)). After successful authentication, afd may be
passed as the second argument to a subsequent mount call (see bind(2)), with the same aname,
as a ticket-of-entry for the user.

If fauth returns −1, the error case, that means the file server does not require authentication for
the connection, and afd should be set to −1 in the call to mount.

It is rare to use fauth directly; more commonly amount (see auth(2)) is used.

SOURCE
/sys/src/libc/9syscall

SEE ALSO
attach(5), auth(2) (particularly amount), authsrv(6), auth(8)

DIAGNOSTICS
Sets errstr.
NAME
Fcall, convS2M, convD2M, convM2S, convM2D, fcallfmt, dirfmt, dirmodefmt, read9pmsg,
statcheck, sizeS2M, sizeD2M – interface to Plan 9 File protocol

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <fcall.h>

uint convS2M(Fcall *f, uchar *ap, uint nap)
uint convD2M(Dir *d, uchar *ap, uint nap)
uint convM2S(uchar *ap, uint nap, Fcall *f)
uint convM2D(uchar *ap, uint nap, Dir *d, char *strs)
int dirfmt(Fmt*)
int fcallfmt(Fmt*)
int dirmodefmt(Fmt*)
int read9pmsg(int fd, uchar *buf, uint nbuf)
int statcheck(uchar *buf, uint nbuf)
uint sizeS2M(Fcall *f)
uint sizeD2M(Dir *d)

DESCRIPTION
These routines convert messages in the machine-independent format of the Plan 9 file protocol,
9P, to and from a more convenient form, an Fcall structure:
#define MAXWELEM 16

typedef
struct Fcall {
  uchar type;
  u32int fid;
  ushort tag;
  union {
    struct {
      u32int msize;          /* Tversion, Rversion */
      char *version;         /* Tversion, Rversion */
    };
    struct {
      ushort oldtag;         /* Tflush */
    };
    struct {
      char *ename;           /* Rerror */
    };
    struct {
      Qid qid;               /* Rattach, Ropen, Rcreate */
      u32int iounit;         /* Ropen, Rcreate */
    };
    struct {
      Qid aqid;              /* Rauth */
    };
    struct {
      u32int afid;           /* Tauth, Tattach */
      char *uname;           /* Tauth, Tattach */
      char *aname;           /* Tauth, Tattach */
    };
  }
};
struct {
    u32int perm;          /* Tcreate */
    char  *name;          /* Tcreate */
    uchar mode;           /* Tcreate, Topen */
};
struct {
    u32int newfid;        /* Twalk */
    ushort nname;         /* Twalk */
    char  *wname[MAXWELEM];/* Twalk */
};
struct {
    ushort nwqid;         /* Rwalk */
    Qid  wqid[MAXWELEM];  /* Rwalk */
};
struct {
    vlong offset;        /* Tread, Twrite */
    u32int count;        /* Tread, Twrite, Rread */
    char  *data;         /* Twrite, Rread */
};
struct {
    ushort nstat;        /* Twstat, Rstat */
    uchar *stat;         /* Twstat, Rstat */
};
} Fcall;

/* these are implemented as macros */
uchar  GBIT8(uchar*)
ushort GBIT16(uchar*)
ulong GBIT32(uchar*)
vlong GBIT64(uchar*)

void PBIT8(uchar*, uchar)
void PBIT16(uchar*, ushort)
void PBIT32(uchar*, ulong)
void PBIT64(uchar*, vlong)

#define BIT8SZ 1
#define BIT16SZ 2
#define BIT32SZ 4
#define BIT64SZ 8

This structure is defined in <fcall.h>. See section 5 for a full description of 9P messages and their encoding. For all message types, the type field of an Fcall holds one of Tversion, Rversion, Tattach, Rattach, etc. (defined in an enumerated type in <fcall.h>). Fid is used by most messages, and tag is used by all messages. The other fields are used selectively by the message types given in comments.

ConvM2S takes a 9P message at ap of length nap, and uses it to fill in Fcall structure f. If the passed message including any data for Twrite and Rread messages is formatted properly, the return value is the number of bytes the message occupied in the buffer ap, which will always be less than or equal to nap; otherwise it is 0. For Twrite and Tread messages, data is set to a pointer into the argument message, not a copy.

ConvS2M does the reverse conversion, turning f into a message starting at ap. The length of the resulting message is returned. For Twrite and Rread messages, count bytes starting at data are copied into the message.

The constant IOHDRSZ is a suitable amount of buffer to reserve for storing the 9P header; the data portion of a Twrite or Rread will be no more than the buffer size negotiated in the Tversion/Rversion exchange, minus IOHDRSZ.
The routine sizeS2M returns the number of bytes required to store the machine-independent representation of the Fcall structure \( f \), including its initial 32-bit size field. In other words, it reports the number of bytes produced by a successful call to convS2M.

Another structure is Dir, used by the routines described in stat(2). ConvM2D converts the machine-independent form starting at \( ap \) into \( d \) and returns the length of the machine-independent encoding. The strings in the returned Dir structure are stored at successive locations starting at \( strs \). Usually \( strs \) will point to storage immediately after the Dir itself. It can also be a nil pointer, in which case the string pointers in the returned Dir are all nil; however, the return value still includes their length.

ConvD2M does the reverse translation, also returning the length of the encoding. If the buffer is too short, the return value will be BIT16SZ and the correct size will be returned in the first BIT16SZ bytes. (If the buffer is less than BIT16SZ, the return value is zero; therefore a correct test for complete packing of the message is that the return value is greater than BIT16SZ). The macro GBIT16 can be used to extract the correct value. The related macros with different sizes retrieve the corresponding-sized quantities. PBIT16 and its brethren place values in messages. With the exception of handling short buffers in convD2M, these macros are not usually needed except by internal routines.

Analogous to sizeS2M, sizeD2M returns the number of bytes required to store the machine-independent representation of the Dir structure \( d \), including its initial 16-bit size field.

The routine statcheck checks whether the \( nbuf \) bytes of \( buf \) contain a validly formatted machine-independent Dir entry suitable as an argument, for example, for the wstat (see stat(2)) system call. It checks that the sizes of all the elements of the entry sum to exactly \( nbuf \), which is a simple but effective test of validity. \( Nbuf \) and \( buf \) should include the second two-byte (16-bit) length field that precedes the entry when formatted in a 9P message (see stat(5)); in other words, \( nbuf \) is 2 plus the sum of the sizes of the entry itself. Statcheck also verifies that the length field has the correct value (that is, \( nbuf-2 \)). It returns 0 for a valid entry and -1 for an incorrectly formatted entry.

Dirfmt, fcallfmt, and dirmodefmt are formatting routines, suitable for fmtinstall(2). They convert Dir*, Fcall*, and long values into string representations of the directory buffer, Fcall buffer, or file mode value. Fcallfmt assumes that dirfmt has been installed with format letter D and dirmodefmt with format letter M.

Read9pmsg calls read(2) multiple times, if necessary, to read an entire 9P message into \( buf \). The return value is 0 for end of file, or -1 for error; it does not return partial messages.

SOURCE
/sys/src/libc/9sys

SEE ALSO
intro(2), 9p(2), stat(2), intro(5)
NAME
fd2path - return file name associated with file descriptor

SYNOPSIS
#include <u.h>
#include <libc.h>
int fd2path(int fd, char *buf, int nbuf)

DESCRIPTION
As described in intro(2), the kernel stores a rooted path name with every open file or directory; typically, it is the name used in the original access of the file. *Fd2path* returns the path name associated with open file descriptor *fd*. Up to *nbuf* bytes of the name are stored in *buf*; if the name is too long, it will be silently truncated at a UTF-8 character boundary. The name is always null-terminated. The return value of *fd2path* will be zero unless an error occurs.

Changes to the underlying name space do not update the path name stored with the file descriptor. Therefore, the path returned by *fd2path* may no longer refer to the same file (or indeed any file) after some component directory or file in the path has been removed, renamed or rebound.

As an example, *getwd(2)* is implemented by opening . and executing *fd2path* on the resulting file descriptor.

SOURCE
/sys/src/libc/9syscall

SEE ALSO
bind(1), ns(1), bind(2), intro(2), getwd(2), proc(3)

DIAGNOSTICS
Sets *errstr*.
NAME
fgetc, getc, getchar, fputc, putc, ungetc, fgetws, gets, fwrite – Stdio input and output

SYNOPSIS
#include <u.h>
#include <stdio.h>
int fgetc(FILE *f)
int getc(FILE *f)
int getchar(void)
int fputc(int c, FILE *f)
int ungetc(int c, FILE *f)
int putc(int c, FILE *f)
int ungetc(int c, FILE *f)
char *fgets(char *s, int n, FILE *f)
char *gets(char *s)
int fprintf(char *s, FILE *f)
int puts(char *s)
long fread(void *ptr, long itemsize, long nitems, FILE *stream)
long fwrite(void *ptr, long itemsize, long nitems, FILE *stream)

DESCRIPTION
The functions described here work on open Stdio streams (see fopen).

fgetc returns an int the next unsigned char from input stream f. If the stream is at end-of-file, the end-of-file indicator for the stream is set and fgetc returns EOF. If a read error occurs, the error indicator for the stream is set and fgetc returns EOF. Getc is like fgetc except that it is implemented as a macro. Getchar is like getc except that it always reads from stdin.

ungetc pushes character c back onto the input stream f. The pushed-back character will be returned by subsequent reads in the reverse order of their pushing. A successful intervening fseek, fsetpos, or rewind on f discards any pushed-back characters for f. One character of push-back is guaranteed. Ungetc returns the character pushed back (converted to unsigned char), or EOF if the operation fails. A successful call to ungetc clears the end-of-file indicator for the stream. The file position indicator for the stream after reading or discarding all pushed-back characters is the same as it was before the characters were pushed back.

fputc writes character c (converted to unsigned char) to output stream f at the position indicated by the position indicator for the stream and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the character is appended to the output stream. Fputc returns the character written or EOF if there was a write error. Putc is like fputc but is implemented as a macro. Putchar is like putc except that it always writes to stdout.

All other input takes place as if characters were read by successive calls to fgetc and all other output takes place as if characters were written by successive calls to fputc.

fgetws reads up to and including the next newline, but not past end-of-file or more than n-1 characters, from stream f into array s. A null character is written immediately after the last character read into the array (if any characters are read at all). Fgetws returns s if successful, otherwise a null pointer. Gets is similar to fgetws except that it always reads from stdin and it discards the terminating newline, if any. Gets does not check for overflow of the receiving array, so its use is deprecated.

fputs writes the string s to stream f, returning EOF if a write error occurred, otherwise a nonnegative value. The terminating null character is not written. Puts is the same, writing to stdout.
Fread reads from the named input stream at most nitems of data of size itemsize and the type of *ptr into a block beginning at ptr. It returns the number of items actually read.

Fwrite appends to the named output stream at most nitems of data of size itemsize and the type of *ptr from a block beginning at ptr. It returns the number of items actually written.

SOURCE
/sys/src/libstdio

SEE ALSO
read(2), fopen(2), bio(2)

BUGS
Stdio does not handle UTF or runes; use Bio instead.
NAME
deflateinit, deflate, deflatezlib, deflateblock, deflatezlibblock, inflateinit, inflate, inflatezlib, inflate-
block, inflatezlibblock,flateerr, mkcrtab, blockcrc, adler32 - deflate compression

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <flate.h>

int deflateinit(void)
int deflate(void *wr, int (*w)(void*, void*, int),
 void *rr, int (*r)(void*, void*, int),
 int level, int debug)
int deflatezlib(void *wr, int (*w)(void*, void*, int),
 void *rr, int (*r)(void*, void*, int),
 int level, int debug)
int deflateblock(uchar *dst, int dsize,
 uchar *src, int ssise,
 int level, int debug)
int deflatezlibblock(uchar *dst, int dsize,
 uchar *src, int ssise,
 int level, int debug)
int inflateinit(void)
int inflate(void *wr, int (*w)(void*, void*, int),
 void *getr, int (*get)(void*))
int inflatezlib(void *wr, int (*w)(void*, void*, int),
 void *getr, int (*get)(void*))
int inflateblock(uchar *dst, int dsize,
 uchar *src, int ssise)
int inflatezlibblock(uchar *dst, int dsize,
 uchar *src, int ssise)
char *flateerr(int error)
ulong *mkcrtab(ulong poly)
ulong blockcrc(ulong *tab, ulong crc, void *buf, int n)
ulong adler32(ulong adler, void *buf, int n)

DESCRIPTION
These routines compress and decompress data using the deflate compression algorithm, which is
used for most gzip, zip, and zlib files.

Deflate compresses input data retrieved by calls to r with arguments rr, an input buffer, and a
count of bytes to read. R should return the number of bytes read; end of input is signaled by
returning zero, an input error by returning a negative number. The compressed output is written
to w with arguments wr, the output data, and the number of bytes to write. W should return the
number of bytes written; writing fewer than the requested number of bytes is an error. Level indicates
the amount of computation deflate should do while compressing the data. Higher levels usually
take more time and produce smaller outputs. Valid values are 1 to 9, inclusive; 6 is a good
compromise. If debug is non-zero, cryptic debugging information is produced on standard error.

Inflate reverses the process, converting compressed data into uncompressed output. Input is
retrieved one byte at a time by calling get with the argument getr. End of input of signaled by
returning a negative value. The uncompressed output is written to w, which has the same interface as for deflate.

Deflateblock and inflateblock operate on blocks of memory but are otherwise similar to deflate and
inflate.
The zlib functions are similar, but operate on files with a zlib header and trailer.

*Deflateinit* or *inflateinit* must be called once before any call to the corresponding routines.

If the above routines fail, they return a negative number indicating the problem. The possible values are *FlateNoMem*, *FlateInputFail*, *FlateOutputFail*, *FlateCorrupted*, and *FlateInternal*. *Flateerr* converts the number into a printable message. *FlateOk* is defined to be zero, the successful return value for *deflateinit*, *deflate*, *deflatezlib*, *inflateinit*, *inflate*, and *inflatezlib*. The block functions return the number of bytes produced when they succeed.

*Mkcrctab* allocates (using *malloc(2)*), initializes, and returns a table for rapid computation of 32 bit CRC values using the polynomial *poly*. *Blockcrc* uses *tab*, a table returned by *mkcrctab*, to update *crc* for the *n* bytes of data in *buf*, and returns the new value. *Crc* should initially be zero. *Blockcrc* pre-conditions and post-conditions *crc* by ones complementation.

*Adler32* updates the Adler 32-bit checksum of the *n* bytes of data in *buf*. The initial value of *adler* (that is, its value after seeing zero bytes) should be 1.

**SOURCE**

/sys/src/libflate
NAME
   fabs, fmod, floor, ceil – absolute value, remainder, floor, ceiling functions

SYNOPSIS
   #include <u.h>
   #include <libc.h>
   double floor(double x)
   double ceil(double x)
   double fabs(double x)
   double fmod(double x, double y)

DESCRIPTION
   Fabs returns the absolute value |x|.
   Floor returns the largest integer not greater than x.
   Ceil returns the smallest integer not less than x.
   Fmod returns x if y is zero, otherwise the number f with the same sign as x, such that
   x = iy + f for some integer i, and |f| < |y|.

SOURCE
   /sys/src/libc/port

SEE ALSO
   abs(2), frexp(2)
NAME
fmtinstall, dofmt, fmitrunc, fmtprint, fmtvprint, fmtstrcpy, fmstrncpy, fmstrndc, fmtfdflush, fmtstrnstr, runefmtstrnstr, runefmtstrndc, errfmt – support for user-defined print formats and output routines

SYNOPSIS
#include <u.h>
#include <libc.h>
typedef struct Fmt Fmt;

struct Fmt{
    uchar runes; /* output buffer is runes or chars? */
    void *start; /* of buffer */
    void *to; /* current place in the buffer */
    void *stop; /* end of the buffer; overwritten if flush fails */
    int (*Flush)(Fmt*); /* called when to == stop */
    void *farg; /* to make flush a closure */
    int fmt; /* num chars formatted so far */
    va_list args; /* args passed to dofmt */
    int nfmt; /* % format Rune */
    ulong flags;
};

enum{
    FmtWidth = 1,
    FmtLeft = FmtWidth << 1,
    FmtPrec = FmtLeft << 1,
    FmtSharp = FmtPrec << 1,
    FmtSpace = FmtSharp << 1,
    FmtSign = FmtSpace << 1,
    FmtZero = FmtSign << 1,
    FmtUnsigned = FmtZero << 1,
    FmtShort = FmtUnsigned << 1,
    FmtLong = FmtShort << 1,
    FmtVLong = FmtLong << 1,
    FmtFlag = FmtVLong << 1
};

int fmtfdinit(Fmt *f, int fd, char *buf, int nbuf);
int fmtfdflush(Fmt *f);
int fmtstrinit(Fmt *f);
char* fmtstrflush(Fmt *f);
char* runefmtstrinit(Fmt *f);
Rune* runefmtstrflush(Fmt *f);
int fmtinstall(int c, int (*fn)(Fmt*));
dofmt(Fmt *f, char *fmt);
dofmt(Fmt*, Rune *fmt);
fmtprint(Fmt *f, char *fmt, ...);
fmtvprint(Fmt *f, char *fmt, va_list v);
fmrune(Fmt *f, int r);
fmruncpy(Fmt *f, char *s);
int fmtrunestrncpy(Fmt *f, Rune *s);
int errfmt(Fmt *f);

DESCRIPTION
The interface described here allows the construction of custom print(2) verbs and output routines. In essence, they provide access to the workings of the formatted print code.

The print(2) suite maintains its state with a data structure called Fmt. A typical call to print(2) or its relatives initializes a Fmt structure, passes it to subsidiary routines to process the output, and finishes by emitting any saved state recorded in the Fmt. The details of the Fmt are unimportant to outside users, except insofar as the general design influences the interface. The Fmt records whether the output is in runes or bytes, the verb being processed, its precision and width, and buffering parameters. Most important, it also records a flush routine that the library will call if a buffer overflows. When printing to a file descriptor, the flush routine will emit saved characters and reset the buffer; when printing to an allocated string, it will resize the string to receive more output. The flush routine is nil when printing to fixed-size buffers. User code need never provide a flush routine; this is done internally by the library.

Custom output routines
To write a custom output routine, such as an error handler that formats and prints custom error messages, the output sequence can be run from outside the library using the routines described here. There are two main cases: output to an open file descriptor and output to a string.

To write to a file descriptor, call fmtfdinit to initialize the local Fmt structure f, giving the file descriptor fd, the buffer buf, and its size nbuf. Then call fmtprint or fmtvprint to generate the output. These behave like fprintf (see print(2)) or vfprintf except that the characters are buffered until fmtfdflush is called and the return value is either 0 or -1. A typical example of this sequence appears in the Examples section.

The same basic sequence applies when outputting to an allocated string: call fmtstrinit to initialize the Fmt, then call fmtprint and fmtvprint to generate the output. Finally, fmtstrflush will return the allocated string, which should be freed after use. To output to a rune string, use runefmtstrinit and runefmtstrflush. Regardless of the output style or type, fmtprint or fmtvprint generates the characters.

Custom format verbs
Fmtinstall is used to install custom verbs and flags labeled by character c, which may be any non-zero Unicode character. Fn should be declared as

```c
int fn(Fmt*)
```

Fp->i is the flag or verb character to cause fn to be called. In fn, fp->width, fp->prec are the width and precision, and fp->flags the decoded flags for the verb (see print(2) for a description of these items). The standard flag values are: FmtSign (+), FmtLeft (-), FmtSpace (' '), FmtSharp (#), FmtComma (,), FmtLong (l), FmtShort (h), FmtUnsigned (u), and FmtVLong (ll). The flag bits FmtWidth and FmtPrec identify whether a width and precision were specified.

Fn is passed a pointer to the Fmt structure recording the state of the output. If fp->i is a verb (rather than a flag), fn should use Fmt->args to fetch its argument from the list, then format it, and return zero. If fp->i is a flag, fn should return one. All interpretation of fp->width, fp->prec, and fp->flags is left up to the conversion routine. Fmtinstall returns 0 if the installation succeeds, -1 if it fails.

Fmtprint and fmtvprint may be called to help prepare output in custom conversion routines. However, these functions clear the width, precision, and flags. Both functions return 0 for success and -1 for failure.

The functions dofmt and dorfmt are the underlying formaters; they use the existing contents of Fmt and should be called only by sophisticated conversion routines. These routines return the number of characters (bytes of UTF or runes) produced.

Some internal functions may be useful to format primitive types. They honor the width, precision and flags as described in print(2). Fmtrune formats a single character x. Fmstrncpy formats a string s; fmtrunestrncpy formats a rune string s. Errfmt formats the system error string. All these routines return zero for successful execution. Conversion routines that call these functions will
work properly regardless of whether the output is bytes or runes.

2c(1) describes the C directive #pragma varargck that can be used to provide type-checking
for custom print verbs and output routines.

EXAMPLES
This function prints an error message with a variable number of arguments and then quits. Compared
to the corresponding example in print(2), this version uses a smaller buffer, will never truncate
the output message, but might generate multiple write system calls to produce its output.

```c
#pragma varargck argpos fatal 1
void
fatal(char *fmt, ...)
{
    Fmt f;
    char buf[64];
    va_list arg;
    fmtfdinit(&f, 1, buf, sizeof buf);
    fmtprint(&f, "fatal: ");
    va_start(arg, fmt);
    fmtvprint(&f, fmt, arg);
    va_end(arg);
    fmtprint(&f, "\n");
    fmtfdflush(&f);
    exits("fatal error");
}
```

This example adds a verb to print complex numbers.

```c
typedef struct {
    double  r, i;
} Complex;
#pragma varargck type"X" Complex
int
Xfmt(Fmt *f)
{
    Complex c;
    c = va_arg(f->args, Complex);
    return fmtprint(f, "(%g,%g)", c.r, c.i);
}
main(...)
{
    Complex x = (Complex){ 1.5, -2.3 }; // colonization
    Xfmtinstall('X', Xfmt);
    print("x = %X\n", x);
}
```

SOURCE
/sys/src/libc/fmt

SEE ALSO
print(2), utf(6), errstr(2)

DIAGNOSTICS
These routines return negative numbers or nil for errors and set errstr.
NAME
fopen, freopen, fdopen, fileno, fclose, fopenr, fopenw, esclose, fflush, setvbuf, setbuf, fgetpos, ftell, fsetpos, fseek, rewind, feof, ferror, clearerr – standard buffered input/output package

SYNOPSIS
#include <stdio.h>
#include <string.h>

FILE *fopen(const char *filename, const char *mode);
FILE *freopen(const char *filename, const char *mode, FILE *f);
FILE *fdopen(int fd, const char *mode);
int fileno(FILE *f);
FILE *sopenr(char *s);
FILE *sopenw(void);
char *sclose(FILE *f);
int fclose(FILE *f);
int fflush(FILE *f);
int setvbuf(FILE *f, char *buf, int type, long size);
void setbuf(FILE *f, char *buf);
int fgetpos(FILE *f, long *pos);
long ftell(FILE *f);
int fsetpos(FILE *f, long *pos);
int fseek(FILE *f, long offset, int whence);
void rewind(FILE *f);
int feof(FILE *f);
int ferror(FILE *f);
void clearerr(FILE *f);

DESCRIPTION
The functions described in this and related pages (fgetc(3), fprintf(3), fscanf(3), and tmpfile(3)) implement the ANSI C buffered I/O package with extensions.

A file with associated buffering is called a stream and is declared to be a pointer to a defined type FILE. fopen(2) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

stdin  standard input file
stdout standard output file
stderr standard error file

A constant pointer NULL designates no stream at all.

Fopen opens the file named by filename and associates a stream with it. Fopen returns a pointer to be used to identify the stream in subsequent operations, or NULL if the open fails. Mode is a character string having one of the following values:
"r"  open for reading
"w"  truncate to zero length or create for writing
"a"  append; open or create for writing at end of file
"r+" open for update (reading and writing)
"w+" truncate to zero length or create for update
"a+" append; open or create for update at end of file

In addition, each of the above strings can have a b somewhere after the first character, meaning 'binary file', but this implementation makes no distinction between binary and text files.
**Fclose** causes the stream pointed to by *f* to be flushed (see below) and does a *close* (see open(2)) on the associated file. It frees any automatically allocated buffer. *Fclose* is called automatically on *exits*(2) for all open streams.

*Freopen* is like open except that it reuses stream pointer *f*. *Freopen* first attempts to close any file associated with *f*; it ignores any errors in that close.

*Fdopen* associates a stream with an open Plan 9 file descriptor.

*Fileno* returns the number of the Plan 9 file descriptor associated with the stream.

*Sopenr* associates a read–only stream with a null–terminated string.

*Sopenw* opens a stream for writing. No file descriptor is associated with the stream; instead, all output is written to the stream buffer.

*Sclose* closes a stream opened with *sopenr* or *sopenw*. It returns a pointer to the 0 terminated buffer associated with the stream.

By default, output to a stream is fully buffered: it is accumulated in a buffer until the buffer is full, and then *write* (see read(2)) is used to write the buffer. An exception is standard error, which is line buffered: output is accumulated in a buffer until a newline is written. Input is also fully buffered by default; this means that *read*(2) is used to fill a buffer as much as it can, and then characters are taken from that buffer until it empties. *Setvbuf* changes the buffering method for file *f* according to *type*: either _IOFBF for fully buffered, _IOLEBF for line buffered, or _IONBF for unbuffered (each character causes a *read* or *write*). If *buf* is supplied, it is used as the buffer and size should be its size; if *buf* is zero, a buffer of the given size is allocated (except for the unbuffered case) using *malloc*(2).

*Setbuf* is an older method for changing buffering. If *buf* is supplied, it changes to fully buffered with the given buffer, which should be of size BUFSIZ (defined in stdio.h). If *buf* is zero, the buffering method changes to unbuffered.

*Flush* flushes the buffer of output stream *f*, delivering any unwritten buffered data to the host file.

There is a *file position indicator* associated with each stream. It starts out pointing at the first character (unless the file is opened with append mode, in which case the indicator is always ignored). The file position indicator is maintained by the reading and writing functions described in *fgetc*(2).

*Fgetpos* stores the current value of the file position indicator for stream *f* in the object pointed to by *pos*. It returns zero on success, nonzero otherwise. *Ftell* returns the current value of the file position indicator. The file position indicator is to be used only as an argument to *fseek*.

*Fsetpos* sets the file position indicator for stream *f* to the value of the object pointed to by *pos*, which shall be a value returned by an earlier call to *fgetpos* on the same stream. It returns zero on success, nonzero otherwise. *Fseek* obtains a new position, measured in characters from the beginning of the file, by adding *offset* to the position specified by *whence*: the beginning of the file if *whence* is SEEK_SET; the current value of the file position indicator for SEEK_CUR; and the end–of–file for SEEK_END. *Rewind* sets the file position indicator to the beginning of the file.

An integer constant EOF is returned upon end of file or error by integer–valued functions that deal with streams. *Feof* returns non–zero if and only if *f* is at its end of file.

*Ferror* returns non–zero if and only if *f* is in the error state. It can get into the error state if a system call failed on the associated file or a memory allocation failed. *Clearerr* takes a stream out of the error state.

**SOURCE**
/sys/src/libstdio

**SEE ALSO**
fprintf(2), fscanf(2), fgetc(2)
open(2), read(2)

**DIAGNOSTICS**
The value EOF is returned uniformly to indicate that a FILE pointer has not been initialized with fopen, input (output) has been attempted on an output (input) stream, or a FILE pointer designates corrupt or otherwise unintelligible FILE data.
Some of these functions set *errstr*.
BUGS

Buffering of output can prevent output data from being seen until long after it is computed – perhaps never, as when an abort occurs between buffer filling and flushing. Buffering of input can cause a process to consume more input than it actually uses. This can cause trouble across `exec(2)`. Buffering may delay the receipt of a write error until a subsequent `stdio` writing, seeking, or file-closing call.

ANSI says that a file can be fully buffered only if the file is not attached to an interactive device. In Plan 9 all are fully buffered except standard error.

`Fdopen`, `fileno`, `sopenr`, `sopenw`, and `sclose` are not ANSI `stdio` functions.

`stdio` offers no support for runes or UTF characters. Unless external compatibility is necessary, use `bio(2)`, which supports UTF and is smaller, faster, and simpler than `stdio`.
NAME
fork, rfork - manipulate process resources

SYNOPSIS
#include <u.h>
#include <libc.h>
int fork(void)
int rfork(int flags)

DESCRIPTION
Forking is the only way new processes are created. The flags argument to rfork selects which resources of the invoking process (parent) are shared by the new process (child) or initialized to their default values. The resources include the file name space, the open file descriptor table (which, when shared, permits processes to open and close files for other processes), the set of environment variables (see env(3)), the note group (the set of processes that receive notes written to a member's notepg file; see proc(3)), the set of rendezvous tags (see rendezvous(2)); and open files. Flags is the logical OR of some subset of

RPFNOC
 If set a new process is created; otherwise changes affect the current process.

RFNOWAIT
 If set, the child process will be dissociated from the parent. Upon exit the child will leave no Waitmsg (see wait(2)) for the parent to collect.

RFNAMEG
 If set, the new process inherits a copy of the parent's name space; otherwise the new process shares the parent's name space. Is mutually exclusive with RFCNAMEG.

RFCNAMEG
 If set, the new process starts with a clean name space. A new name space must be built from a mount of an open file descriptor. Is mutually exclusive with RFNAMEG.

RFNOMNT
 If set, subsequent mounts into the new name space and dereferencing of pathnames starting with # are disallowed.

RFENVG
 If set, the environment variables are copied; otherwise the two processes share environment variables. Is mutually exclusive with RFCENVG.

RFCENVG
 If set, the new process starts with an empty environment. Is mutually exclusive with RFENVG.

RFNOTE
 Each process is a member of a group of processes that all receive notes when a note is written to any of their notepg files (see proc(3)). The group of a new process is by default the same as its parent, but if RFNOTE is set (regardless of RFPROC), the process becomes the first in a new group, isolated from previous processes.

RFFDG
 If set, the invoker's file descriptor table (see intro(2)) is copied; otherwise the two processes share a single table.

RFCFDG
 If set, the new process starts with a clean file descriptor table. Is mutually exclusive with RFFDG.

RFOREND
 If set, the process will be unable to rendezvous(2) with any of its ancestors; its children will, however, be able to rendezvous with it. In effect, RFOREND makes the process the first in a group of processes that share a space for rendezvous tags.

RFMEM
 If set, the child and the parent will share data and bss segments. Otherwise, the child inherits a copy of those segments. Other segment types, in particular stack segments, will be unaffected. May be set only with RFPROC.

File descriptors in a shared file descriptor table are kept open until either they are explicitly closed or all processes sharing the table exit.

If RFPROC is set, the value returned in the parent process is the process id of the child process; the value returned in the child is zero. Without RFPROC, the return value is zero. Process ids range from 1 to the maximum integer (int) value. Rfork will sleep, if necessary, until required process resources are available.

Fork is just a call of rfork(RFFDG|RFOREND|RFPROC).

SOURCE
/sys/src/libc/9syscall
/sys/src/libc/9sys/fork.c
SEE ALSO
    intro(2), proc(3),

DIAGNOSTICS
    These functions set errstr.
NAME
fprintf, printf, sprintf, snprintf, vsprintf, vfprintf, vsprintf, vsnprintf – print formatted output

SYNOPSIS
#include <u.h>
#include <stdio.h>
int fprintf(FILE *f, char *format, ...)
int printf(char *format, ...)
int snprintf(char *s, char *format, ...)
int vsprintf(char *format, va_list args)
int fprintf(FILE *f, char *format, va_list args)
int vsprintf(char *s, char *format, va_list args)
int vsnprintf(char *s, int n, char *format, va_list args)

DESCRIPTION
fprintf places output on the named output stream f (see fopen(2)). printf places output on the standard output stream stdout. sprintf places output followed by the null character (\0) in consecutive bytes starting at s; it is the user’s responsibility to ensure that enough storage is available. snprintf is like sprintf but writes at most n bytes (including the null character) into s. Vfprintf, vprintf, vsprintf, and vsnprintf are the same, except the args argument is the argument list of the calling function, and the effect is as if the calling function’s argument list from that point on is passed to the printf routines.

Each function returns the number of characters transmitted (not including the \0 in the case of sprintf and friends), or a negative value if an output error was encountered.

These functions convert, format, and print their trailing arguments under control of a format string. The format contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more arguments. The results are undefined if there are arguments of the wrong type or too few arguments for the format. If the format is exhausted while arguments remain, the excess are ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:
   Zero or more flags, which modify the meaning of the conversion specification.
   An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left adjustment, described later, has been given) to the field width.
   An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversions, the number of digits to appear after the decimal point for the e, E, and F conversions, the maximum number of significant digits for the g and G conversions, or the maximum number of characters to be written from a string in s conversion. The precision takes the form of a period (.) followed by an optional decimal integer; if the integer is omitted, it is treated as zero.
   An optional h specifying that a following d, i, o, u, x or X conversion specifier applies to a short int or unsigned short argument (the argument will have been promoted according to the integral promotions, and its value shall be converted to short or unsigned short before printing); an optional h specifying that a following n conversion specifier applies to a pointer to a short argument; an optional l (ell) specifying that a following d, i, o, u, x, or X conversion character applies to a long or unsigned long argument; an optional l specifying that a following n conversion specifier applies to a pointer to a long int argument; or an optional L specifying that a following e, E, f, g, or G conversion specifier applies to a long double argument. If an h, l, or L appears with any other conversion specifier, the behavior is undefined.
A character that indicates the type of conversion to be applied.

A field width or precision, or both, may be indicated by an asterisk (*) instead of a digit string. In this case, an int arg supplies the field width or precision. The arguments specifying field width or precision, or both, shall appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a − flag followed by a positive field width. A negative precision is taken as if it were missing.

The flag characters and their meanings are:

- The result of the conversion is left-justified within the field.
+ The result of a signed conversion always begins with a sign (+ or −).
blank If the first character of a signed conversion is not a sign, or a signed conversion results in no characters, a blank is prefixed to the result. This implies that if the blank and + flags both appear, the blank flag is ignored.
# The result is to be converted to an “alternate form.” For o conversion, it increases the precision to force the first digit of the result to be a zero. For x or X conversion, a non-zero result has 0x or 0X prefixed to it. For e, E, f, g, and G conversions, the result always contains a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For g and G conversions, trailing zeros are not be removed from the result as they normally are. For other conversions, the behavior is undefined.
0 For d, i, o, u, x, X, e, E, f, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad the field width; no space padding is performed. If the 0 and − flags both appear, the 0 flag will be ignored. For d, i, o, u, x, and X conversions, if a precision is specified, the 0 flag will be ignored. For other conversions, the behavior is undefined.

The conversion characters and their meanings are:

d,o,u,x,X
The integer arg is converted to signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x or X); the letters abcdedef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

f The double argument is converted to decimal notation in the style [-]ddd..ddd, where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly 0, no decimal point appears.
e,E
The double argument is converted in the style [-]d.ddde±dd, where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, it is taken as 6; if the precision is zero, no decimal point appears. The E format code produces a number with E instead of e introducing the exponent. The exponent always contains at least two digits.
g,G
The double argument is printed in style f or e (or in style E in the case of a G conversion specifier), with the precision specifying the number of significant digits. If an explicit precision is zero, it is taken as 1. The style used depends on the value converted: style e is used only if the exponent resulting from the conversion is less than −4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a decimal point appears only if it is followed by a digit.
c
The int argument is converted to an unsigned char, and the resulting character is written.
s
The argument is taken to be a string (character pointer) and characters from the string are printed until a null character (\0) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A zero value for the argument yields undefined results.
P
The void* argument is printed in an implementation-defined way (for Plan 9: the address as hexadecimal number).
n
The argument shall be a pointer to an integer into which is written the number of characters written to the output stream so far by this call to printf. No argument is
converted.
% Print a %; no argument is converted.

If a conversion specification is invalid, the behavior is undefined.

If any argument is, or points to, a union or an aggregate (except for an array of character type using %s conversion, or a pointer cast to be a pointer to void using %P conversion), the behavior is undefined.

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

SOURCE
/sys/src/libstdio

SEE ALSO
fopen(2), fscanf(2), print(2)

BUGS
There is no way to print a wide character (rune); use print(2) or bio(2).
NAME
frinit, frsetrects, frinittick, frclear, frcharofpt, frptofchar, frinsert, frdelete, frselect, frtick, frselectpaint, frdrawsel, frdrawsel0, frgetmouse – frames of text

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <thread.h>
#include <mouse.h>
#include <frame.h>

void frinit(Frame *f, Rectangle r, Font *ft, Image *b, Image **cols)
void frsetrects(Frame *f, Rectangle r, Image *b)
void frinittick(Frame *f)
void frclear(Frame *f, int resize)
ulong frcharofpt(Frame *f, Point pt)
Point frptofchar(Frame *f, ulong p)
void frinsert(Frame *f, Rune *r0, Rune *r1, ulong p)
int frdelete(Frame *f, ulong p0, ulong p1)
void frselect(Frame *f, Mousectl *m)
void frtick(Frame *f, Point pt, int up)
void frselectpaint(Frame *f, Point p0, Point p1, Image *col)
void frdrawsel(Frame *f, Point pt0, ulong p0, ulong p1, int highlighted)
void frdrawsel0(Frame *f, Point pt0, ulong p0, ulong p1, Image *back, Image *text)

enum{
  BACK,
  HIGH,
  BORD,
  TEXT,
  HTEXT,
  NCOL
};

DESCRIPTION
This library supports frames of editable text in a single font on raster displays, such as in sam(1) and ria(1). Frames may hold any character except NUL (0). Long lines are folded and tabs are at fixed intervals.

The user-visible data structure, a Frame, is defined in <frame.h>:

typedef struct Frame Frame;
struct Frame{
  Font *font; /* of chars in the frame */
  Display *display; /* on which frame appears */
  Image *b; /* on which frame appears */
  Image *cols[NCOL]; /* text and background colors */
  Rectangle r; /* in which text appears */
  Rectangle entire; /* of full frame */
  Frbox *box;
  ulong p0, p1; /* selection */
  ushort nbox, nalloc;
  ushort maxtab; /* max size of tab, in pixels */
ushort  nchars;  /* # runes in frame */
ushort  nlines;  /* # lines with text */
ushort  maxlines;  /* total # lines in frame */
ushort  lastlinefull;  /* last line fills frame */
ushort  modified;  /* changed since frselect() */
Image  *tick;  /* typing tick */
Image  *tickback;  /* saved image under tick */
tint  ticked;  /* flag: is tick onscreen? */
};

Frbox is an internal type and is not used by the interface. P0 and p1 may be changed by the
application provided the selection routines are called afterwards to maintain a consistent display.
Maxtab determines the size of tab stops. Frinit sets it to 8 times the width of a 0 (zero) character
in the font; it may be changed before any text is added to the frame. The other elements of the
structure are maintained by the library and should not be modified directly.

The text within frames is not directly addressable; instead frames are designed to work alongside
another structure that holds the text. The typical application is to display a section of a longer
document such as a text file or terminal session. Usually the program will keep its own copy of the
text in the window (probably as an array of Runes) and pass components of this text to the frame
routines to display the visible portion. Only the text that is visible is held by the Frame; the appli-
cation must check maxlines, nlines, and lastlinefull to determine, for example, whether new text needs to be appended at the end of the Frame after calling frdelete (q.v.).

There are no routines in the library to allocate Frames; instead the interface assumes that
Frames will be components of larger structures. Frinit prepares the Frame f so characters
drawn in it will appear in the single Font ft. It then calls frsetrects and frinittick to initialize the
geometry for the Frame. The Image b is where the Frame is to be drawn; Rectangle r
defines the limit of the portion of the Image the text will occupy. The Image pointer may be
null, allowing the other routines to be called to maintain the associated data structure in, for exam-
ple, an obscured window.

The array of Images cols sets the colors in which text and borders will be drawn. The back-
ground of the frame will be drawn in cols[BACK]; the background of highlighted text in
cols[HIGH]; borders and scroll bar in cols[BORD]; regular text in cols[TEXT]; and high-
lighted text in cols[HTEXT].

Frclear frees the internal structures associated with f, permitting another frinit or frsetrects on the
Frame. It does not clear the associated display. If f is to be deallocated, the associated Font
and Image must be freed separately. The resize argument should be non-zero if the frame is
to be redrawn with a different font; otherwise the frame will maintain some data structures associ-
ated with the font.

To resize a Frame, use frclear and frinit and then frinsert (q.v.) to recreate the display. If a
Frame is being moved but not resized, that is, if the shape of its containing rectangle is
unchanged, it is sufficient to use draw(2) to copy the containing rectangle from the old to the new
location and then call frsetrects to establish the new geometry. (It is unnecessary to call frinittick
unless the font size has changed.) No redrawing is necessary.

Frames hold text as runes, not as bytes. Frptofchar returns the location of the upper left corner
of the p'th rune, starting from 0, in the Frame f. If f holds fewer than p runes, frptofchar returns
the location of the upper right corner of the last character in f. Frcharofpt is the inverse: it returns
the index of the closest rune whose image's upper left corner is up and to the left of pt.

Frinsert inserts into Frame f starting at rune index p the runes between r0 and r1. If a NUL (0)
character is inserted, chaos will ensue. Tabs and newlines are handled by the library, but all other
characters, including control characters, are just displayed. For example, backspaces are printed;
to erase a character, use frdelete.

Frdelete deletes from the Frame the text between p0 and p1; p1 points at the first rune beyond the
deletion.

Frselect tracks the mouse to select a contiguous string of text in the Frame. When called, a
mouse button is typically down. Frselect will return when the button state has changed (some but-
tons may still be down) and will set f->p0 and f->p1 to the selected range of text.
Programs that wish to manage the selection themselves have several routines to help. They involve the maintenance of the 'tick', the vertical line indicating a null selection between characters, and the colored region representing a non-null selection. Frtick draws (if up is non-zero) or removes (if up is zero) the tick at the screen position indicated by pt. Frdrawsel repaints a section of the frame, delimited by character positions p0 and p1, either with plain background or entirely highlighted, according to the flag highlighted, managing the tick appropriately. The point pt0 is the geometrical location of p0 on the screen; like all of the selection-helper routines’ Point arguments, it must be a value generated by frptofchar. Frdrawsel0 is a lower-level routine, taking as arguments a background color, back, and text color, text. It assumes that the tick is being handled (removed beforehand, replaced afterwards, as required) by its caller. Frselectpaint uses a solid color, col, to paint a region of the frame defined by the Points p0 and p1.

SOURCE
/sys/src/libframe

SEE ALSO
  graphics(2), draw(2), cachechars(2).
NAME
frexp, ldexp, modf – split into mantissa and exponent

SYNOPSIS
#include <u.h>
#include <libc.h>
double frexp(double value, int *eptr)
double ldexp(double value, int exp)
double modf(double value, double *iptr)

DESCRIPTION
Frexp returns the mantissa of value and stores the exponent indirectly through eptr, so that value = frexp(value) \times 2^{eptr}

Ldexp returns the quantity value \times 2^{exp}.

Modf returns the signed fractional part of value and stores the integer part indirectly through iptr.

SOURCE
/sys/src/libc/port/frexp.c

SEE ALSO
intro(2)

DIAGNOSTICS
Ldexp returns 0 for underflow and the appropriately signed infinity for overflow.
NAME
fscanf, scanf, sscanf, vfscanf - scan formatted input

SYNOPSIS
#include <u.h>
#include <stdio.h>

int fscanf(FILE *f, char *format, ...)
int scanf(char *format, ...)
int sscanf(char *s, char *format, ...)
int vfscanf(FILE *stream, char *format, char *args)

DESCRIPTION
Fscanf reads from the named input stream f (see fopen(2)) under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored.

Scanf and sscanf are the same, but they read from stdin and the character string s, respectively. Vfscanf is like scanf, except the args argument is a pointer to an argument in an argument list of the calling function and the effect is as if the calling function’s argument list from that point on is passed to the scanf routines.

The format is composed of zero or more directives: one or more white-space characters; an ordinary character (not %); or a conversion specification. Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

An optional assignment–suppressing character *.

An optional decimal integer that specifies the maximum field width.

An optional h, l (ell) or L indicating the size of the receiving object. The conversion specifiers d, i, and n shall be preceded by h if the corresponding argument is a pointer to short rather than a pointer to int, or by l if it is a pointer to long. Similarly, the conversion specifiers o, u, and x shall be preceded by h if the corresponding argument is a pointer to unsigned short rather than a pointer to unsigned, or by l if it is a pointer to unsigned long. Finally, the conversion specifiers e, f, and g shall be preceded by l if the corresponding argument is a pointer to double rather than a pointer to float, or by L if it is a pointer to long double. If an h, l, or L appears with any other conversion specifier, the behavior is undefined.

A character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

Fscanf executes each directive of the format in turn. If a directive fails, as detailed below, fscanf returns. Failures are described as input failures (due to the unavailability of input), or matching failures (due to inappropriate input).

A directive composed of white space is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary character is executed by reading the next character from the stream. If it differs from the one comprising the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

Input white–space characters (as specified by isspace, see ctype(2)) are skipped, unless the specification includes a [, c, or n specifier.

An input item is read from the stream, unless the specification includes an n specifier. An input item is defined as the longest sequence of input characters (up to any specified maximum field width) which is an initial subsequence of a matching sequence. The first character, if any, after the
input item remains unread. If the length of the input item is zero, the execution of the directive fails: this condition is a matching failure, unless an error prevented input from the stream, in which case it is an input failure.

Except in the case of a % specifier, the input item (or, in the case of a %n directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a *, the result of the conversion is placed in the object pointed to by the first argument following the format argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The following conversion specifiers are valid:

d Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the *strtol* (see *atof(2)*) function with 10 for the base argument. The corresponding argument shall be a pointer to *int*.

i Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the *strtol* function with 0 for the base argument. The corresponding argument shall be a pointer to *int*.

o Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the *strtoul* (see *atof(2)*) function with 8 for the base argument. The corresponding argument shall be a pointer to *unsigned int*.

u Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the *strtoul* function with 10 for the base argument. The corresponding argument shall be a pointer to *unsigned int*.

x Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the *strtoul* function with 16 for the base argument. The corresponding argument shall be a pointer to *unsigned int*.

e, f, g Matches an optionally signed floating-point number, whose format is the same as expected for the subject string of the *strtod* (see *atof(2)*) function. The corresponding argument shall be a pointer to *float*.

s Matches a sequence of non-white-space characters. The corresponding argument shall be a pointer to the initial character of an array large enough to accept the sequence and a terminating NUL (0) character, which will be added automatically.

[ Matches a nonempty sequence of characters from a set of expected characters (the scanset). The corresponding argument shall be a pointer to the initial character of an array large enough to accept the sequence and a terminating NUL character, which will be added automatically. The conversion specifier includes all subsequent characters in the format string, up to and including the matching right brace (]). The characters between the brackets (the scanlist) comprise the scanset, unless the character after the left bracket is a circumflex (^), in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. As a special case, if the conversion specifier begins with [ ] or [ ^], the right bracket character is in the scanlist and the next right bracket character is the matching right bracket that ends the specification. If a – character is in the scanlist and is not the first, nor the second where the first character is a ^, nor the last character, the behavior is implementation-defined (in Plan 9: the scanlist includes all characters in the ASCII (sic) range between the two characters on either side of the –).

c Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument shall be a pointer to the initial character of an array large enough to accept the sequence. No NUL character is added.

P Matches an implementation-defined set of sequences, which should be the same as the set of sequences that may be produced by the %P conversion of the *fprintf(2)* function (in Plan 9, a hexadecimal number). The corresponding argument shall be a pointer to a pointer to void. The interpretation of the input item is implementation defined; however, for any input item other than a value converted earlier during the same program execution, the behavior of the %P conversion is undefined.
n  No input is consumed. The corresponding argument shall be a pointer to integer into which is written the number of characters read from the input stream so far by this call to \textit{fscanf}. Execution of a \%n directive does not increment the assignment count returned at the completion of \textit{fscanf}.

\%
  Matches a single \%; no conversion or assignment occurs. The complete conversion specification shall be \%.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers E, G, and X are also valid and behave the same as, respectively, e, g, and x.

If end–of–file is encountered during input, conversion is terminated. If end–of–file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space (including newline characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the \%n directive.

The return value from \textit{fscanf} is the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure. However, if an input failure occurs before any conversion, EOF is returned.

\textbf{SOURCE}
/sys/src/libstdio

\textbf{SEE ALSO}
\textit{fopen}(2), \textit{fgetc}(2)

\textbf{BUGS}
  Does not know about UTF.
NAME
  fversion – initialize 9P connection and negotiate version

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  int  fversion(int fd, int bufsize, char *version, int nversion)

DESCRIPTION
  Fversion is used to initialize the 9P connection represented by fd and
to negotiate the version of the protocol to be used.

  The bufsize determines the size of the I/O buffer used to stage 9P
requests to the server, subject to the constraints of the server itself. The
version is a text string that represents the highest version level the
protocol will support. The version will be overwritten with the negotiated, possibly
lower, version of the protocol. The return value of fversion is the length of the
returned version string; the value of nversion is therefore not the length of the
version string presented to the system call, but the total length of the buffer to accept
the final result, in the manner of a read system call.

  Default values of zero for bufsize and the empty string for version will negotiate sensible defaults
for the connection. If version is the empty string, nversion must still be large enough to receive
the returned version string.

  The interpretation of the version strings is defined in version(5).

  It is rare to use fversion directly; usually the default negotiation performed by the kernel during
mount (see bind(2)) or even more commonly amount (see auth(2)) is sufficient.

SOURCE
  /sys/src/libc/9syscall

SEE ALSO
  intro(5), version(5), fauth(2).

DIAGNOSTICS
  Sets errstr.
NAME
getcallerpc - fetch return PC of current function

SYNOPSIS
#include <u.h>
#include <libc.h>
uintptr getcallerpc(void *firstarg)

DESCRIPTION
Getcallerpc is a portable way to discover the PC to which the current function will return. Firstarg should be a pointer to the first argument to the function in question.

EXAMPLE
void
printpc(int arg)
{
    print("Called from %p\n", getcallerpc(&arg));
}

void
main(int argc, char *argv[])
{
    printpc(0);
    printpc(0);
    printpc(0);
}

SOURCE
/sys/src/libc/$objtype/getcallerpc.[cs]

BUGS
The firstarg parameter should not be necessary.
NAME
getenv, putenv – access environment variables

SYNOPSIS
#include <u.h>
#include <libc.h>
char* getenv(char *name)
int putenv(char *name, char *val)

DESCRIPTION
Getenv reads the contents of /env/ name (see env(3)) into memory allocated with malloc(2), 0-terminates it, and returns a pointer to that area. If no file exists, 0 is returned.

Putenv creates the file /env/ name and writes the string val to it. The terminating 0 is not written. If the file value cannot be written, -1 is returned.

SOURCE
/sys/src/libc/9sys

SEE ALSO
env(3)

DIAGNOSTICS
Sets errstr.
NAME
getfcr, setfcr, getfsr, setfsr - control floating point

SYNOPSIS
#include <u.h>
#include <libc.h>
ulong getfcr(void)
void setfcr(ulong fcr)
ulong getfsr(void)
void setfsr(ulong fsr)

DESCRIPTION
These routines provide a fairly portable interface to control the rounding and exception characteristics of IEEE 754 floating point units. In effect, they define a pair of pseudo-registers, the floating point control register, fcr, which affects rounding, precision, and exceptions, and the floating point status register, fsr, which holds the accrued exception bits. Each register has a get routine to retrieve its value, a set routine to modify it, and macros that identify its contents.

The fcr contains bits that, when set, halt execution upon exceptions: FPINEX (enable inexact exceptions), FPOVFL (enable overflow exceptions), FPUNFL (enable underflow exceptions), FPZDIV (enable zero divide exceptions), and FPINVAL (enable invalid operation exceptions). Rounding is controlled by installing in fcr, under mask FPRMASK, one of the values FPRNR (round to nearest), FPRZ (round towards zero), FPRPINF (round towards positive infinity), and FPRNINF (round towards negative infinity). Precision is controlled by installing in fcr, under mask FPPMASK, one of the values FPPEXT (extended precision), FPPSGL (single precision), and FPPDBL (double precision).

The fsr holds the accrued exception bits FPAINEX, FPAOVFL, FPAUNFL, FPZDIV, and FPINVAL, corresponding to the fcr bits without the A in the name.

Not all machines support all modes. If the corresponding mask is zero, the machine does not support the rounding or precision modes. On some machines it is not possible to clear selective accrued exception bits; a setfsr clears them all. The exception bits defined here work on all architectures. Where possible, the initial state is equivalent to

        setfcr(FPPDBL|FPRNR|FPINVAL|FPZDIV|FPOVFL);

However, this may vary between architectures: the default is to provide what the hardware does most efficiently. Use these routines if you need guaranteed behavior. Also, gradual underflow is not available on some machines.

EXAMPLE
To enable overflow traps and make sure registers are rounded to double precision (for example on the MC68020, where the internal registers are 80 bits long):

        setfcr((getfcr() & ~FPPMASK) | FPPDBL | FPOVFL);

SOURCE
/sys/src/libc/$objtype/getfcr.s
NAME
getfields, gettokens, tokenize - break a string into fields

SYNOPSIS
#include <u.h>
#include <libc.h>
int getfields(char *str, char **args, int maxargs, int multiflag, char *delims)
int gettokens(char *str, char **args, int maxargs, char *delims)
int tokenize(char *str, char **args, int maxargs)

DESCRIPTION
Getfields places into the array args pointers to the first maxargs fields of the null terminated UTF string str. Delimiters between these fields are set to null.

Fields are substrings of str whose definition depends on the value of multiflag. If multiflag is zero, adjacent fields are separated by exactly one delimiter. For example

getfields("#alice#bob##charles###", arg, 3, 0, ":");

yields three substrings: null-string, alice, and bob##charles##. If the multiflag argument is not zero, a field is a non-empty string of non-delimiters. For example

getfields("#alice#bob##charles###", arg, 3, 1, ":");

yields the three substrings: alice, bob, and charles##.

Getfields returns the number of fields pointed to.

Gettokens is the same as getfields with multiflag non-zero, except that fields may be quoted using single quotes, in the manner of rc(1). Any such quotes remain in the resulting args. See quote(2) for related quote-handling software.

Tokenize is similar to gettokens with delims set to ":\t\r\n ", except that quotes are interpreted but do not appear in the resulting args.

SOURCE
/sys/src/libc/port/tokenize.c

SEE ALSO
strtok in strcat(2), quote(2).
NAME
g getpid, getppid - get process ids

SYNOPSIS
#include <u.h>
#include <libc.h>
int getpid(void)
int getppid(void)

DESCRIPTION
Getpid reads /dev/pid (see cons(3)) and converts it to get the process id of the current process, a number guaranteed to be unique among all running processes on the machine executing getpid.

Getppid reads /dev/ppid (see cons(3)) and converts it to get the id of the parent of the current process.

SOURCE
/sys/src/libc/9sys

SEE ALSO
intro(2), exec(2), cons(3), proc(3)

DIAGNOSTICS
Returns 0 and sets errstr if unsuccessful.
NAME
  getuser, sysname - get user or system name

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  char* getuser(void)
  char* sysname(void)

DESCRIPTION
  Getuser returns a pointer to static data which contains the null-terminated name of the user who
  owns the current process. Getuser reads /dev/user to find the name.

  Sysname provides the same service for the file #c/sysname, which contains the name of the
  machine. Unlike getuser, sysname caches the string, reading the file only once.

SOURCE
  /sys/src/libc/port/getuser.c

SEE ALSO
  intro(2), cons(3)
NAME
getwd – get current directory

SYNOPSIS
#include <u.h>
#include <libc.h>
char* getwd(char *buf, int size)

DESCRIPTION
Getwd fills buf with a null-terminated string representing the current directory and returns buf.
Getwd places no more than size bytes in the buffer provided.

SOURCE
/sys/src/libc/9sys/getwd.c

SEE ALSO
pwd(1), fd2path(2)

DIAGNOSTICS
On error, zero is returned. Errstr(2) may be consulted for more information.

BUGS
Although the name returned by getwd is guaranteed to be the path used to reach the directory, if
the name space has changed underfoot, the name may be incorrect.
NAME
Display, Point, Rectangle, Cursor, initdraw, geninitdraw, drawerror, initdisplay, closedisplay, getdef-
font, getwindow, gengetwindow, flushimage, bufimage, lockdisplay, unlockdisplay, openfont,
buildfont, freefont, Pfmt, Rfmt, strtochan, chantostr, chantodepth - interactive graphics

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <cursor.h>
int initdraw(void (*errfun)(Display*, char*), char *font,
             char *label)
int geninitdraw(char *devdir, void(*errfun)(Display*, char*),
               char *font, char *label, char *mousedir, char *windir,
               int ref)
int newwindow(char *str)
void drawerror(Display *d, char *msg)
Display* initdisplay(char *devdir, char *win, void(*errfun)(Display*, char*))
void closedisplay(Display *d)
Subfont* getdefont(Display *d)
int flushimage(Display *d, int vis)
uchar* bufimage(Display *d, int n)
void lockdisplay(Display *d)
void unlockdisplay(Display *d)
int getwindow(Display *d, int ref)
int gengetwindow(Display *d, char *winname,
              Image **ip, Screen **sp, int ref)
Font* openfont(Display *d, char *name)
Font* buildfont(Display *d, char *desc, char *name)
void freefont(Font *f)
int Pfmt(Fmt*)
int Rfmt(Fmt*)
ulong strtochan(char *s)
char* chantostr(char *s, ulong chan)
int chantodepth(ulong chan)
extern Display *display
extern Image *screen
extern Screen *_screen
extern Font *font

DESCRIPTION
A Display structure represents a connection to the graphics device, draw(3), holding all graph-
ics resources associated with the connection, including in particular raster image data in use by
the client program. The structure is defined (in part) as:

typedef
struct Display
{
...
**void (*error)(Display*, char*);**

... Image *black; Image *white; Image *opaque; Image *transparent; Image *image; Font *defaultfont; Subfont*defaultsubfont; ...}

A Point is a location in an Image (see below and draw(2)), such as the display, and is defined as:

typedef
struct Point {
    int x;
    int y;
} Point;

The coordinate system has x increasing to the right and y increasing down.

A Rectangle is a rectangular area in an image.

typedef
struct Rectangle {
    Point min; /* upper left */
    Point max; /* lower right */
} Rectangle;

By definition, min.x≤max.x and min.y≤max.y. By convention, the right (maximum x) and bottom (maximum y) edges are excluded from the represented rectangle, so abutting rectangles have no points in common. Thus, max contains the coordinates of the first point beyond the rectangle.

The Image data structure is defined in draw(2).

A Font is a set of character images, indexed by runes (see utf(6)). The images are organized into Subfonts, each containing the images for a small, contiguous set of runes. The detailed format of these data structures, which are described in detail in cachechars(2), is immaterial for most applications. Font and Subfont structures contain two interrelated fields: ascent, the distance from the top of the highest character (actually the top of the image holding all the characters) to the baseline, and height, the distance from the top of the highest character to the bottom of the lowest character (and hence, the interline spacing). See cachechars(2) for more details. Buildfont parses the font description in the buffer desc, returning a Font* pointer that can be used by string (see draw(2)) to draw characters from the font. Openfont does the same, but reads the description from the named file. Freefont frees a font. The convention for naming font files is:

```
/lib/font/bit/name/range.size.font
```

where size is approximately the height in pixels of the lower case letters (without ascenders or descendents). Range gives some indication of which characters will be available: for example ascii, latin1, euro, or unicode. Euro includes most European languages, punctuation marks, the International Phonetic Alphabet, etc., but no Oriental languages. Unicode includes every character for which appropriate-sized images exist on the system.

A Cursor is defined:

typedef struct
Cursor {
    Point offset;
    uchar clr[2*16];
    uchar set[2*16];
} Cursor;
The arrays are arranged in rows, two bytes per row, left to right in big-endian order to give 16 rows of 16 bits each. A cursor is displayed on the screen by adding $\texttt{off}$ set to the current mouse position, using $\texttt{clr}$ as a mask to draw white at the pixels where $\texttt{clr}$ is one, and then drawing black at the pixels where set is one. Setcursor and moveto (see mouse(2)) and esetcursor and emoveto (see event(2)) change the cursor image and its location on the screen.

The routine initdraw connects to the display; it returns -1 if it fails and sets the error string. Initdraw sets up the global variables display (the Display structure representing the connection), screen (an Image representing the display memory itself or, if $\texttt{rio}$ is running, the client's window), and $\texttt{font}$ (the default font for text). The arguments to initdraw include a label, which is written to /dev/Label if non-nil so that it can be used to identify the window when hidden (see $\texttt{rio}$(1)). The font is created by reading the named font file. If $\texttt{font}$ is null, initdraw reads the file named in the environment variable $\texttt{font}$; if $\texttt{font}$ is not set, it imports the default (usually minimal) font from the operating system. The global $\texttt{font}$ will be set to point to the result. The $\texttt{font}$ argument is a graphics error function to call in the event of a fatal error in the library; it must never return. Its arguments are the display pointer and an error string. If errfun is nil, the library provides a default, called drawerror. Another effect of initdraw is that it installs print(2) formats Pfmt and Rfmt as %P and %R for printing Points and Rectangles.

The geninitdraw function provides a less automated way to establish a connection, for programs that wish to connect to multiple displays. Devdir is the name of the directory containing the device files for the display (if nil, default /dev); errfun, font, and label are as in initdraw; mousedir and windir are the directories holding the mouse and winname files; and ref specifies the refresh function to be used to create the window, if running under $\texttt{rio}$(1) (see window(2)).

The function newwindow may be called before initdraw or geninitdraw to cause the program to occupy a newly created window rather than take over the one in which it is running when it starts. The str argument, if non-null, is concatenated to the string "new " that is used to create the window (see $\texttt{rio}$(4)). For example, newwindow("-hide -dy 100") will cause the program to run in a newly created, hidden window 100 pixels high.

Initdisplay is part of geninitdraw; it sets up the display structures but does not allocate any fonts or call getwindow. The arguments are similar to those of initdraw; win names the directory, default /dev, in which the files associated with the window reside. Closedisplay disconnects the display and frees the associated data structures. Getdefont builds a SubFont structure from in-core data describing a default subfont. None of these routines is needed by most programs, since initdraw calls them as needed.

The data structures associated with the display must be protected in a multi-process program, because they assume only one process will be using them at a time. Multi-process programs should set $\texttt{display-}\rightarrow\texttt{locking}$ to 1, to notify the library to use a locking protocol for its own accesses, and call lockdisplay and unlockdisplay around any calls to the graphics library that will cause messages to be sent to the display device. Initdraw and geninitdraw initialize the display to the locked state.

Getwindow returns a pointer to the window associated with the application; it is called automatically by initdraw to establish the screen pointer but must be called after each resizing of the window to restore the library's connection to the window. If $\texttt{rio}$ is not running, it returns display-image; otherwise it negotiates with $\texttt{rio}$ by looking in /dev/winname to find the name of the window and opening it using namedimage (see allocimage(2)). The resulting window will be created using the refresh method ref (see window(2)); this should almost always be Refnnone because $\texttt{rio}$ provides backing store for the window.

Getwindow overwrites the global variables screen, a pointer to the Image defining the window (or the overall display, if no window system is running); and _screen, a pointer to the Screen representing the root of the window's hierarchy. (See window(2). The overloading of the screen word is an unfortunate historical accident.) Getwindow assigns that screen point to the portion of the window inside the border; sophisticated clients may use _screen to make further subwindows. Programs desiring multiple independent windows may use the mechanisms of $\texttt{rio}$(4) to create more windows (usually by a fresh mount of the window sytem in a directory other than /dev), then use gengetwindow to connect to them. Gengetwindow's extra arguments are the full path of the window's winname file and pointers to be overwritten with the values of the 'global' Image and Screen variables for the new window.
The graphics functions described in `draw(2)`, `allocimage(2)`, `cachechars(2)`, and `subfont(2)` are implemented by writing commands to files under `/dev/draw` (see `draw(3)`); the writes are buffered, so the functions may not take effect immediately. `Flushimage` flushes the buffer, doing all pending graphics operations. If `vis` is non-zero, any changes are also copied from the 'soft screen' (if any) in the driver to the visible frame buffer. The various allocation routines in the library flush automatically, as does the event package (see `event(2)`); most programs do not need to call `flushimage`. It returns -1 on error.

`Bufimage` is used to allocate space for $n$ bytes in the display buffer. It is used by all the graphics routines to send messages to the display.

The functions `strtochan` and `chantostr` convert between the channel descriptor strings used by `image(6)` and the internal ulong representation used by the graphics protocol (see `draw(3)`'s b message). Chantostr writes at most nine bytes into the buffer pointed at by $s$ and returns $s$ on success, 0 on failure. Chantodepth returns the number of bits per pixel used by the format specified by `chan`. Both chantodepth and strtochan return 0 when presented with bad input.

**EXAMPLES**

To reconnect to the window after a resize event,

```c
if(getwindow(display, Refnone) < 0)
    sysfatal("resize failed: %r");
```

To create and set up a new `rio(1)` window,

```c
Image *screen2;
Screen *_screen2;

srvwsys = getenv("wsys");
if(srvwsys == nil)
    sysfatal("can't find $wsys: %r");
rfork(RENAMEG); /* keep mount of rio private */

fd = open(srvwsys, ORDWR);
if(fd < 0)
    sysfatal("can't open $wsys: %r");

/* mount creates window; see `rio(4)` */
if(mount(fd, -1, "/tmp", MREPL, "new -dx 300-dy 200") < 0)
    sysfatal("can't mount new window: %r");
if(gengetwindow(display, "/tmp/winname",
    &screen2, _screen2, Refnone) < 0)
    sysfatal("resize failed: %r");

/* now open /tmp/cons, /tmp/mouse */
...```

**FILES**

```
/lib/font/bit directory of fonts
```

**SOURCE**

```
/sys/src/libdraw
```

**SEE ALSO**

`rio(1)`, `addpt(2)`, `allocimage(2)`, `cachechars(2)`, `subfont(2)`, `draw(2)`, `event(2)`, `frame(2)`, `print(2)`, `window(2)`, `draw(3)`, `rio(4)`, `image(6)`, `font(6)`

**DIAGNOSTICS**

An error function may call `errstr(2)` for further diagnostics.

**BUGS**

The names `clr` and `set` in the Cursor structure are reminders of an archaic color map and might be more appropriately called white and black.
NAME

parsetml, printitems, validitems, freeitems, freedocinfo, dimenkind, dimenspec, targetid, target-
name, fromStr, toStr – HTML parser

SYNOPSIS

#include <u.h>
#include <libc.h>
#include <html.h>
Item* parsetml(uchar* data, int datalen, Rune* src, int mtype,
   int chset, Docinfo** pdi)
void printitems(Item* items, char* msg)
int validitems(Item* items)
void freeitems(Item* items)
void freedocinfo(Docinfo* d)
int dimenkind(Dimen d)
int dimenspec(Dimen d)
int targetid(Rune* s)
Rune* targetname(int targid)
uchar* fromStr(Rune* buf, int n, int chset)
Rune* toStr(uchar* buf, int n, int chset)

DESCRIPTION

This library implements a parser for HTML 4.0 documents. The parsed HTML is converted into an
intermediate representation that describes how the formatted HTML should be laid out.

Parsetml parses an entire HTML document contained in the buffer data and having length
datalen. The URL of the document should be passed in as src. Mtype is the media type of the doc-
ument, which should be either TextHtml or TextPlain. The character set of the document is described
in chset, which can be one of US_Ascii, ISO_8859_1, UTF_8 or Unicode. The return value is a linked list of Item structures, described in detail below. As a side effect, *pdi is
set to point to a newly created Docinfo structure, containing information pertaining to the entire
document.

The library expects two allocation routines to be provided by the caller, emalloc and
eremalloc. These routines are analogous to the standard malloc and realloc routines, except
that they should not return if the memory allocation fails. In addition, emalloc is required to
zero the memory.

For debugging purposes, printitems may be called to display the contents of an item list; individual
items may be printed using the %I print verb, installed on the first call to parsetml.
validitems traverses the item list, checking that all of the pointers are valid. It returns 1 is every-
things is ok, and 0 if an error was found. Normally, one would not call these routines directly.
Instead, one sets the global variable dbgl and the library calls them automatically. One can
also set warn, to cause the library to print a warning whenever it finds a problem with the input
document, and dbglex, to print debugging information in the lexer.

When an item list is finished with, it should be freed with freeitems. Then, freedocinfo should be
called on the pointer returned in *pdi.

Dimenkind and dimenspec are provided to interpret the Dimen type, as described in the section
Dimension Specifications.

Frame target names are mapped to integer ids via a global, permanent mapping. To find the value
for a given name, call targetid, which allocates a new id if the name hasn’t been seen before.
The name of a given, known id may be retrieved using targetname. The library predefines FTtop,
FTSelf, FTParent and FTBlank.

The library handles all text as Unicode strings (type Rune*). Character set conversion is provided
by fromStr and toStr. FromStr takes n Unicode characters from buf and converts them to the
character set described by chset. ToStr takes n bytes from buf, interpreted as belonging to char-
cacter set chset, and converts them to a Unicode string. Both routines null-terminate the result,
and use emalloc to allocate space for it.

Items
The return value of parsehtml is a linked list of variant structures, with the generic portion
described by the following definition:

typedef struct Item Item;
struct Item
{
    Item* next;
    int width;
    int height;
    int ascent;
    int anchorid;
    int state;
    Genattr* genattr;
    int tag;
};

The field next points to the successor in the linked list of items, while width, height, and
ascent are intended for use by the caller as part of the layout process. Anchorid, if non-zero,
gives the integer id assigned by the parser to the anchor that this item is in (see section Anchors).
State is a collection of flags and values described as follows:

enum
{
    IIfbrk = 0x80000000,
    IIfbrksp = 0x40000000,
    IIfnobrk = 0x20000000,
    IIfcleft = 0x10000000,
    IIfcright = 0x08000000,
    IIfwrap = 0x04000000,
    IIfhang = 0x02000000,
    IIfjust = 0x01000000,
    IIfcjust = 0x00800000,
    IIfmap = 0x00400000,
    IIfindentshift = 8,
    IIfindentmask = (255<<IIfindentshift),
    IIfhangmask = 255
};

IIfbrk is set if a break is to be forced before placing this item. IIfbrksp is set if a 1 line space
should be added to the break (in which case IIfbrk is also set). IIfnobrk is set if a break is not
permitted before the item. IIfcleft is set if left floats should be cleared (that is, if the list of
pending left floats should be placed) before this item is placed, and IIfcright is set for right
floats. In both cases, IIfbrk is also set. IIfwrap is set if the line containing this item is allowed to
wrap. IIfhang is set if this item hangs into the left indent. IIfjust is set if the line containing
this item should be right justified, and IIfcjust is set for center justified lines. IIfmap is used to
indicate that an image is a server-side map. The low 8 bits, represented by IIfhangmask,
indicate the current hang into left indent, in tenths of a tabstop. The next 8 bits, represented by
IIfindentmask and IIfindentshift, indicate the current indent in tab stops.

The field genattr is an optional pointer to an auxiliary structure, described in the section
Generic Attributes.

Finally, tag describes which variant type this item has. It can have one of the values Itexttag,
Iruletag, Iimagetag, Iformfieldtag, Itabletag, Ifloattag or Ispacertag.
For each of these values, there is an additional structure defined, which includes Item as an
unnamed initial substructure, and then defines additional fields.

Items of type Itexttag represent a piece of text, using the following structure:
struct Itext
{
    Item;
    Rune* s;
    int fnt;
    int fg;
    uchar voff;
    uchar ul;
};

Here s is a null-terminated Unicode string of the actual characters making up this text item, fnt is the font number (described in the section Font Numbers), and fg is the RGB encoded color for the text. Voff measures the vertical offset from the baseline; subtract Voffbias to get the actual value (negative values represent a displacement down the page). The field ul is the underline style: ULnone if no underline, ULunder for conventional underline, and ULmid for strike-through.

Items of type Iruletag represent a horizontal rule, as follows:

struct Irule
{
    Item;
    uchar align;
    uchar nosshade;
    int size;
    Dimen wspec;
};

Here align is the alignment specification (described in the corresponding section), nosshade is set if the rule should not be shaded, size is the height of the rule (as set by the size attribute), and wspec is the desired width (see section Dimension Specifications).

Items of type Iimagetag describe embedded images, for which the following structure is defined:

struct Iimage
{
    Item;
    Rune* imsrc;
    int imwidth;
    int imheight;
    Rune* altrep;
    Map* map;
    int ctclid;
    uchar align;
    uchar hspace;
    uchar vspace;
    uchar border;
    Iimage* nextimage;
};

Here imsrc is the URL of the image source, imwidth and imheight, if non-zero, contain the specified width and height for the image, and altrep is the text to use as an alternative to the image, if the image is not displayed. Map, if set, points to a structure describing an associated client-side image map. Ctclid is reserved for use by the application, for handling animated images. Align encodes the alignment specification of the image. Hspace contains the number of pixels to pad the image with on either side, and Vspace the padding above and below. Border is the width of the border to draw around the image. Nextimage points to the next image in the document (the head of this list is Docinfo.images).

For items of type Iformfieldtag, the following structure is defined:

struct Iformfield
{
    Item;
}
Formfield* formfield;
);
This adds a single field, formfield, which points to a structure describing a field in a form, described in section Forms.
For items of type Itabletag, the following structure is defined:
struct Itable
{
  Item;
  Table* table;
};
Table points to a structure describing the table, described in the section Tables.
For items of type Ifloattag, the following structure is defined:
struct Ifloat
{
  Item;
  Item* item;
  int x;
  int y;
  uchar side;
  uchar infloats;
  Ifloat* nextfloat;
};
The item points to a single item (either a table or an image) that floats (the text of the document flows around it), and side indicates the margin that this float sticks to; it is either ALleft or ALright. x and y are reserved for use by the caller; these are typically used for the coordinates of the top of the float. Infloats is used by the caller to keep track of whether it has placed the float. Nextfloat is used by the caller to link together all of the floats that it has placed.
For items of type Ispacertag, the following structure is defined:
struct Ispacertag
{
  Item;
  int spkind;
};
Spkind encodes the kind of spacer, and may be one of ISPnull (zero height and width), ISPvline (takes on height and ascent of the current font), ISPhspace (has the width of a space in the current font) and ISPgeneral (for all other purposes, such as between markers and lists).

Generic Attributes
The genattr field of an item, if non-nil, points to a structure that holds the values of attributes not specific to any particular item type, as they occur on a wide variety of underlying HTML tags. The structure is as follows:
typedef struct Genattr Genattr;
struct Genattr
{
  Rune* id;
  Rune* class;
  Rune* style;
  Rune* title;
  SEvent* events;
};
Fields id, class, style and title, when non-nil, contain values of correspondingly named attributes of the HTML tag associated with this item. Events is a linked list of events (with corresponding scripted actions) associated with the item:
typedef struct SEvent SEvent;
struct SEvent
{
    SEvent* next;
    int type;
    Rune* script;
};

Here, next points to the next event in the list, type is one of SEonblur, SEonchange, SEonclick, SEondblclick, SEonfocus, SEonkeypress, SEonkeyup, SEonload, SEonmousedown, SEonmousemove, SEonmouseout, SEonmouseover, SEonmouseup, SEonreset, SEonselect, SEonsubmit or SEonunload, and script is the text of the associated script.

**Dimension Specifications**

Some structures include a dimension specification, used where a number can be followed by a % or a * to indicate percentage of total or relative weight. This is encoded using the following structure:

typedef struct Dimen Dimen;
struct Dimen
{
    int kindspec;
};

Separate kind and spec values are extracted using `dimenkind` and `dimenspec`. `Dimenkind` returns one of Dnone, Dpixels, Dpercent or Drelative. Dnone means that no dimension was specified. In all other cases, `dimenspec` should be called to find the absolute number of pixels, the percentage of total, or the relative weight.

**Background Specifications**

It is possible to set the background of the entire document, and also for some parts of the document (such as tables). This is encoded as follows:

typedef struct Background Background;
struct Background
{
    Rune* image;
    int color;
};

Image, if non-nil, is the URL of an image to use as the background. If this is nil, `color` is used instead, as the RGB value for a solid fill color.

**Alignment Specifications**

Certain items have alignment specifiers taken from the following enumerated type:

enum
{
    ALnone = 0, ALleft, ALcenter, ALright, ALjustify,
    ALchar, ALtop, ALmiddle, ALbottom, ALbaseline
};

These values correspond to the various alignment types named in the HTML 4.0 standard. If an item has an alignment of ALleft or ALright, the library automatically encapsulates it inside a float item.

Tables, and the various rows, columns and cells within them, have a more complex alignment specification, composed of separate vertical and horizontal alignments:

typedef struct Align Align;
struct Align
{
    uchar halign;
    uchar valign;
};
Halign can be one of ALnone, ALleft, ALcenter, ALright, A$	extbf{L}$justify or A$	extbf{L}$char. Valign can be one of ALnone, ALmiddle, ALbottom, ALtop or ALbaseline.

Font Numbers
Text items have an associated font number (the fnt field), which is encoded as style$^*$NumSize+size. Here, style is one of FntR, FntI, FntB or FntT, for roman, italic, bold and typewriter font styles, respectively, and size is Tiny, Small, Normal, Large or VeryLarge. The total number of possible font numbers is NumFnt, and the default font number is DefFnt (which is roman style, normal size).

Document Info
Global information about an HTML page is stored in the following structure:

```c
typedef struct Docinfo Docinfo;
struct Docinfo
{
    // stuff from HTTP headers, doc head, and body tag
    Rune* src;
    Rune* base;
    Rune* doctitle;
    Background background;
    Iimage* backgrounditem;
    int text;
    int link;
    int vlink;
    int alink;
    int target;
    int chset;
    int mediatype;
    int scripttype;
    int hasscripts;
    Rune* refresh;
    Kidinfo* kidinfo;
    int frameid;

    // info needed to respond to user actions
    Anchor* anchors;
    DestAnchor* dests;
    Form* forms;
    Table* tables;
    Map* maps;
    Iimage* images;
};
```

Src gives the URL of the original source of the document, and base is the base URL. Doctitle is the document's title, as set by a <title> element. Background is as described in the section Background Specifications, and backgrounditem is set to be an image item for the document's background image (if given as a URL), or else nil. Text gives the default foreground text color of the document, link the unvisited hyperlink color, vlink the visited hyperlink color, and alink the color for highlighting hyperlinks (all in 24-bit RGB format). Target is the default target frame id. Chset and mediatype are as for the chset and mtype parameters to parsehtml. Scripttype is the type of any scripts contained in the document, and is always TextJavascript. Hasscripts is set if the document contains any scripts. Scripting is currently unsupported. Refresh is the contents of a <meta http-equiv=Refresh ...> tag, if any. Kidinfo is set if this document is a frameset (see section Frames). Frameid is this document's frame id.

Anchors is a list of hyperlinks contained in the document, and dests is a list of hyperlink destinations within the page (see the following section for details). Forms, tables and maps are lists of the various forms, tables and client-side maps contained in the document, as described in subsequent sections. Images is a list of all the image items in the document.
Anchors

The library builds two lists for all of the <a> elements (anchors) in a document. Each anchor is assigned a unique anchor id within the document. For anchors which are hyperlinks (the href attribute was supplied), the following structure is defined:

typedef struct Anchor Anchor;
struct Anchor {
    Anchor* next;
    int index;
    Rune* name;
    Rune* href;
    int target;
};

Next points to the next anchor in the list (the head of this list is Docinfo.anchors). Index is the anchor id; each item within this hyperlink is tagged with this value in its anchorid field. Name and href are the values of the correspondingly named attributes of the anchor (in particular, href is the URL to go to). Target is the value of the target attribute (if provided) converted to a frame id.

Destinations within the document (anchors with the name attribute set) are held in the Docinfo.dests list, using the following structure:

typedef struct DestAnchor DestAnchor;
struct DestAnchor {
    DestAnchor* next;
    int index;
    Rune* name;
    Item* item;
};

Next is the next element of the list, index is the anchor id, name is the value of the name attribute, and item is points to the item within the parsed document that should be considered to be the destination.

Forms

Any forms within a document are kept in a list, headed by Docinfo.forms. The elements of this list are as follows:

typedef struct Form Form;
struct Form {
    Form* next;
    int formid;
    Rune* name;
    Rune* action;
    int target;
    int method;
    int nfields;
    Formfield* fields;
};

Next points to the next form in the list. Formid is a serial number for the form within the document. Name is the value of the form’s name or id attribute. Action is the value of any action attribute. Target is the value of the target attribute (if any) converted to a frame target id. Method is one of HGet or HPost. Nfields is the number of fields in the form, and fields is a linked list of the actual fields.

The individual fields in a form are described by the following structure:

typedef struct Formfield Formfield;
struct Formfield {

Formfield* next;
int ftype;
int fieldid;
Form* form;
Rune* name;
Rune* value;
int size;
int maxlength;
int rows;
int cols;
uchar flags;
Option* options;
Item* image;
int fieldid;
int ftype;
Formfield* next;
int border;
Dimen width;
int ncell;
Tablecell*** grid;
Align align;

Here, next points to the next field in the list. Ftype is the type of the field, which can be one of Ftext, Fpassword, Fcheckbox, Fradio, Fsubmit, Fhidden, Fimage, Freset, Ffile, Fbutton, Fselect or Ftextarea. Fieldid is a serial number for the field within the form. Form points back to the form containing this field. Name, value, size, maxlength, rows and cols each contain the values of corresponding attributes of the field, if present. Flags contains per-field flags, of which FFchecked and FFmultiple are defined. Image is only used for fields of type Fimage; it points to an image item containing the image to be displayed. Ctidid is reserved for use by the caller, typically to store a unique id of an associated control used to implement the field. Events is the same as the corresponding field of the generic attributes associated with the item containing this field. Options is only used by fields of type Fselect; it consists of a list of possible options that may be selected for that field, using the following structure:

typedef struct Option Option;
struct Option
{
    Option* next;
    int selected;
    Rune* value;
    Rune* display;
};

Next points to the next element of the list. Selected is set if this option is to be displayed initially. Value is the value to send when the form is submitted if this option is selected. Display is the string to display on the screen for this option.

Tables
The library builds a list of all the tables in the document, headed by Docinfo.tables. Each element of this list has the following format:

typedef struct Table Table;
struct Table
{
    Table* next;
    int tableid;
    Tablerow* rows;
    int nrow;
    Tablecol* cols;
    int ncol;
    Tablecell* cells;
    int ncell;
    Tablecell*** grid;
    Align align;
    Dimen width;
    int border;
Next points to the next element in the list of tables. Tableid is a serial number for the table within the document. Rows is an array of row specifications (described below) and nrow is the number of elements in this array. Similarly, cols is an array of column specifications, and ncol the size of this array. Cells is a list of all cells within the table (structure described below) and ncell is the number of elements in this list. Note that a cell may span multiple rows and/or columns, thus ncell may be smaller than nrow*ncol. Grid is a two-dimensional array of cells within the table; the cell at row i and column j is Table.grid[i][j]. A cell that spans multiple rows and/or columns will be referenced by grid multiple times, however it will only occur once in cells. Align gives the alignment specification for the entire table, and width gives the requested width as a dimension specification. Border, cellspacing and cellpadding give the values of the corresponding attributes for the table, and background gives the requested background for the table. Caption is a linked list of items to be displayed as the caption of the table, either above or below depending on whether caption.place is ATop or ABottom. Most of the remaining fields are reserved for use by the caller, except tabletok, which is reserved for internal use. The type Lay is not defined by the library; the caller can provide its own definition.

The Tablecol structure is defined for use by the caller. The library ensures that the correct number of these is allocated, but leaves them blank. The fields are as follows:

typedef struct Tablecol Tablecol;
struct Tablecol
{
  int    width;
  Align  align;
  Point  pos;
};

The rows in the table are specified as follows:

typedef struct Tablerow Tablerow;
struct Tablerow
{
  Tablerow*  next;
  Tablecell* cells;
  int        height;
  int        ascent;
  Align      align;
  Background background;
  Point      pos;
  uchar      flags;
};

Next is only used during parsing; it should be ignored by the caller. Cells provides a list of all the cells in a row, linked through their nextinrow fields (see below). Height, ascent and pos are reserved for use by the caller. Align is the alignment specification for the row, and background is the background to use, if specified. Flags is used by the parser; ignore this field.
The individual cells of the table are described as follows:

typedef struct Tablecell Tablecell;
struct Tablecell {
    Tablecell* next;
    Tablecell* nextinrow;
    Item* content;
    Lay* lay;
    int colspan;
    int rowspan;
    Align align;
    uchar flags;
    int width;
    int height;
    Background background;
    int minw;
    int maxw;
    int ascent;
    int row;
    int col;
    Point pos;
};

Next is used to link together the list of all cells within a table (Table.cells), whereas
nextinrow is used to link together all the cells within a single row (Tablerow.cells).
CellId provides a serial number for the cell within the table. Content is a linked list of the
items to be laid out within the cell. Lay is reserved for the user to describe how these items have
been laid out. Rowspan and colspan are the number of rows and columns spanned by this
cell, respectively. Align is the alignment specification for the cell. Flags is some combination
of TFparsing, TFnowrap and TFisth or'd together. Here TFparsing is used internally by
the parser, and should be ignored. TFnowrap means that the contents of the cell should not be
wrapped if they don't fit the available width, rather, the table should be expanded if need be (this
is set when the nowrap attribute is supplied). TFisth means that the cell was created by the
<th> element (rather than the <td> element), indicating that it is a header cell rather than a data
cell. Wspec provides a suggested width as a dimension specification, and hspec provides a sug-
gested height in pixels. Background gives a background specification for the individual cell.
Minw, maxw, ascent and pos are reserved for use by the caller during layout. Row and col
give the indices of the row and column of the top left-hand corner of the cell within the table grid.

Client-side Maps
The library builds a list of client-side maps, headed by Docinfo.maps, and having the following
structure:

typedef struct Map Map;  
struct Map {
    Map* next;
    Rune* name;
    Area* areas;
};

Next points to the next element in the list, name is the name of the map (use to bind it to an
image), and areas is a list of the areas within the image that comprise the map, using the follow-
ing structure:

typedef struct Area Area; 
struct Area {
    Area* next;
    int shape;
    Rune* href;
}
int target;
Dimen* coords;
int ncoords;
}

Next points to the next element in the map's list of areas. Shape describes the shape of the area, and is one of SHrect, SHcircle or SHpoly. Href is the URL associated with this area in its role as a hypertext link, and target is the target frame it should be loaded in. Coords is an array of coordinates for the shape, and ncoords is the size of this array (number of elements).

**Frames**

If the Docinfo.kidinfo field is set, the document is a frameset. In this case, it is typical for parsehtml to return nil, as a document which is a frameset should have no actual items that need to be laid out (such will appear only in subsidiary documents). It is possible that items will be returned by a malformed document; the caller should check for this and free any such items.

The Kidinfo structure itself reflects the fact that framesets can be nested within a document. If is defined as follows:

```c
typedef struct Kidinfo Kidinfo;
struct Kidinfo
{
    Kidinfo* next;
    int isframeset;

    // fields for "frame"
    Rune* src;
    Rune* name;
    int marginw;
    int marginh;
    int framebd;
    int flags;

    // fields for "frameset"
    Dimen* rows;
    int nrows;
    Dimen* cols;
    int ncols;
    Kidinfo* kidinfos;
    Kidinfo* nextframeset;
};
```

Next is only used if this structure is part of a containing frameset; it points to the next element in the list of children of that frameset. Isframeset is set when this structure represents a frameset; if clear, it is an individual frame.

Some fields are used only for framesets. Rows is an array of dimension specifications for rows in the frameset, and nrows is the length of this array. Cols is the corresponding array for columns, of length ncols. Kidinfos points to a list of components contained within this frameset, each of which may be a frameset or a frame. Nextframeset is only used during parsing, and should be ignored.

The remaining fields are used if the structure describes a frame, not a frameset. Src provides the URL for the document that should be initially loaded into this frame. Note that this may be a relative URL, in which case it should be interpreted using the containing document's URL as the base. Name gives the name of the frame, typically supplied via a name attribute in the HTML. If no name was given, the library allocates one. Marginw, marginh and framebd are the values of the marginwidth, marginheight and frameborder attributes, respectively. Flags can contain some combination of the following: FRnoresize (the frame had the noresize attribute set, and the user should not be allowed to resize it), FRnoscroll (the frame should not have any scroll bars), FRhscroll (the frame should have a horizontal scroll bar), FRvscroll (the frame should have a vertical scroll bar), FRhscrollauto (the frame should be automatically given a horizontal scroll bar if its contents would not otherwise fit), and FRvscrollauto (the frame gets a vertical scrollbar only if required).
SOURCE
/sys/src/libhtml

SEE ALSO
fmt(1)

BUGS
The entire HTML document must be loaded into memory before any of it can be parsed.
NAME
HConnect, HContent, HContents, HETag, HFields, Hio, Htmlesc, HttpHead, HttpReq, HRange,
HSPairs, hmydomain, hversion, htmlesc, halloc, hbodypush, hbuflen, hcheckcontent, hclose,
hdate2sec, hdatefmt, hfail, hflush, hgetc, hgethead, hinit, hiserror, hload, hlower, hmkcontent,
hmkfields, hmkmimeboundary, hmkpairs, hmoved, hokheaders, hparserequest, hparsereq, hprint, hputc, hreadbuf, hredirected, hreqcleanup, hrevhfields, hrevespairs, hstrdup,
http11, httpfmt, httpunesc, hunallowed, hungetc, hunload, hurlfmt, hurlunesc, hvprint, hwrite,
hxferenc,
- routines for creating an http server

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <httpd.h>
typedef struct HConnect HConnect;
typedef struct HContent HContent;
typedef struct HContents HContents;
typedef struct HETag HETag;
typedef struct HFields HFields;
typedef struct Hio Hio;
typedef struct Htmlesc Htmlesc;
typedef struct HttpHead HttpHead;
typedef struct HttpReq HttpReq;
typedef struct HRange HRange;
typedef struct HSPairs HSPairs;
typedef struct Bin Bin;

struct Htmlesc
{
    char    *name;
    Rune     value;
};

struct HContent
{
    HContent     *next;
    char            *generic;
    char            *specific;
    float           q;       /* desirability of this kind of file */
    int             mxb;     /* max uchars until worthless */
};

struct HContents
{
    HContent     *type;
    HContent     *encoding;
};

/*
 * generic http header with a list of tokens,
 * each with an optional list of parameters
 */
struct HFields
{
    char            *s;
    HSPairs         *params;
    HFields         *next;
struct HSPairs
{
    char *s;
    char *t;
    HSPairs *next;
};

/*
 * list of pairs a strings
 * used for tag=val pairs for a search or form submission,
 * and attribute=value pairs in headers.
 */

struct HRange
{
    int suffix; /* is this a suffix request? */
    ulong start;
    ulong stop; /* ~0UL -> not given */
    HRange *next;
};

/*
 * byte ranges within a file
 */

struct HETag
{
    char *etag;
    int weak;
    HETag *next;
};

/*
 * list of http/1.1 entity tags
 */

enum
{
    Hnone,
    Hread,
    Hend,
    Hwrite,
    Herr,

    Hsize = HBufSize
};

struct Hio
{
    Hio *hh; /* next lower layer Hio, or nil if */
    int fd; /* associated file descriptor */
    ulong seek; /* of start */
    uchar state; /* state of the file */
    uchar xferenc; /* chunked transfer encoding status */
    uchar *pos; /* current position in the buffer */
    uchar *stop; /* last character active in the bu
uchar *start; /* start of data buffer */
ulong bodylen; /* remaining length of message body */
uchar buf[Hsize+32];

/*
 * request line
 */
struct HttpReq
{
    char *meth;
    char *uri;
    char *urihost;
    char *search;
    int vermaj;
    int vermin;
};

/*
 * header lines
 */
struct HttpHead
{
    int closeit; /* http1.1 close connection after */
    uchar persist; /* http/1.1 requests a persistent */
    uchar expectcont; /* expect a 100-continue */
    uchar expectother; /* expect anything else; should reject */
    ulong contlen; /* if != ~0UL, length of included */
    HFields *transenc; /* if present, encoding of included */
    char *client;
    char *host;
    HContent *okencode;
    HContent *oklang;
    HContent *oktype;
    HContent *okchar;
    ulong ifmodsince;
    ulong ifunmodsince;
    ulong ifrangedate;
    HETag *ifmatch;
    HETag *ifnomatch;
    HETag *ifrangeetag;
    HRange *range;
    char *authuser; /* authorization info */
    char *authpass;

    /*
     * experimental headers
     */
    int fresh_thresh;
    int fresh_have;
};

/*
 * all of the state for a particular connection
 */
struct HConnect
{
    void *private; /* for the library clients */
void
  (*replog)(HConnect*, char*, ...); /* called when reply sent */

HttpReq  req;
HttpHead  head;

Bin        *bin;
ulong      reqtime; /* time at start of request */
char       xferbuf[HBufSize]; /* buffer for making up or transferring
uchar      header[HBufSize + 2]; /* room for \n\0 */
uchar      *hpos;
uchar      *hstop;
Hio        hin;
Hio        hout;

/* configuration for all connections within the server */
extern char *hmydomain;
extern char *hversion;
extern Htmlesc htmlesc[];

void        *halloc(HConnect *c, ulong size);
Hio        *hbufpush(Hio *hh, ulong len, HFields *te);
int         hbuflen(Hio *h, void *p);
int         hcheckcontent(HContent*, HContent*, char*, int);
void        hclose(Hio*);
ulong       hdate2sec(char*);
int         hdatefmt(Fmt*);
int         hfail(HConnect*, int, ...);
int         hflush(Hio*);
int         hgetc(Hio*);
int         hgethead(HConnect *c, int many);
int         hinit(Hio*, int, int);
int         hiserror(Hio *h);
int         hload(Hio*, char*);
char        *hlower(char*);
HContent    *hmkcontent(HConnect *c, char *generic, char *specific, HContent *next);
HFields    *hmkhfields(HConnect *c, char *s, HSPairs *p, HFields *next);
char        *hmkmimeboundary(HConnect *c);
HSPairs    *hmkspairs(HConnect *c, char *s, char *t, HSPairs *next);
int         hmoved(HConnect *c, char *uri);
void        hokheaders(HConnect *c);
int         hparseheaders(HConnect*, int timeout);
HSPairs    *hparsequery(HConnect *c, char *search);
int         hparsequery(HConnect *c, int timeout);
int         hprint(Hio*, char*, ...);
int         hputc(Hio*, int);
void        *hreadbuf(Hio *h, void *vsave);
int         hredirected(HConnect *c, char *how, char *uri);
void        hreqcleanup(HConnect *c);
HFields    *hrevhfields(HFields *hf);
HSPairs    *hrevspairs(HSPairs *sp);
char        *hstrdup(HConnect *c, char *s);
int         http11(HConnect*);
int         httpfmt(Fmt*);
char        *httpunesc(HConnect *c, char *s);
int         hunallowed(HConnect *, char *allowed);
int hungetc(Hio *h);
char *hunload(Hio *);
int hurlfmt(Fmt *);
char *hurlunesc(HConnect *c, char *s);
int hvprint(Hio *, char *, va_list);
int hwrite(Hio *, void *, int);
int hxferenc(Hio *, int);

DESCRIPTION
For now, look at the source, or httpd(8).

SOURCE
/sys/src/libhttpd

SEE ALSO
bin(2)

BUGS
This is a rough implementation and many details are going to change.
NAME
  hypot - Euclidean distance

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  double hypot(double x, double y)

DESCRIPTION
  Hypot returns
    \sqrt{x^2 + y^2}
  taking precautions against unwarranted overflows.

SOURCE
  /sys/src/libc/port/hypot.c
NAME
Intmap, allocmap, freemap, insertkey, caninsertkey, lookupkey, deletekey – integer to data struc-
ture maps

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <fcall.h>
#include <thread.h>
#include <9p.h>

Intmap* allocmap(void (*inc)(void*))
void freemap(Intmap *map, void (*dec)(void*))
void* insertkey(Intmap *map, ulong key, void *val)
int caninsertkey(Intmap *map, ulong key, void *val)
void* lookupkey(Intmap *map, ulong key)
void* deletekey(Intmap *map, ulong key)

DESCRIPTION
An Intmap is an arbitrary mapping from integers to pointers. Allocmap creates a new map, and
freemap destroys it. The inc function is called each time a new pointer is added to the map; similar-
ly, dec is called on each pointer left in the map when it is being freed. Typically these functions
maintain reference counts. New entries are added to the map by calling insertkey, which will
return the previous value associated with the given key, or zero if there was no previous value.
Caninsertkey is like insertkey but only inserts val if there is no current mapping. It returns 1 if val
was inserted, 0 otherwise. Lookupkey returns the pointer associated with key, or zero if there is
no such pointer. Deletekey removes the entry for id from the map, returning the associated
pointer, if any.

Concurrent access to Intmaps is safe, moderated via a QLock stored in the Intmap structure.
In anticipation of the storage of reference-counted structures, an increment function inc may be
specified at map creation time. Lookupkey calls inc (if non-zero) on pointers before returning
them. If the reference count adjustments were left to the caller (and thus not protected by the
lock), it would be possible to accidentally reclaim a structure if, for example, it was deleted from
the map and its reference count decremented between the return of insertkey and the external
increment. Insertkey and caninsertkey do not call inc when inserting val into the map, nor do
insertkey or deletekey call inc when returning old map entries. The rationale is that calling an
insertion function transfers responsibility for the reference to the map, and responsibility is given
back via the return value of deletekey or the next insertkey.

Intmaps are used by the 9p library to implement Fidpools and Reqpools.

SOURCE
/sys/src/lib9p/intmap.c

SEE ALSO
9p(2), 9pfid(2).
NAME
closeioproc, iocall, ioclose, iointerrupt, iodial, ioopen, ioproc, ioread, ioreadn, iowrite – slave I/O processes for threaded programs

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <thread.h>

typedef struct Ioproc Ioproc;

Ioproc* ioproc(void);
int iopen(Ioproc *io, char *file, int omode);
int ioclose(Ioproc *io, int fd);
long ioread(Ioproc *io, int fd, void *a, long n);
long ioreadn(Ioproc *io, int fd, void *a, long n);
long iowrite(Ioproc *io, int fd, void *a, long n);
int iodial(Ioproc *io, char *addr, char *local, char *dir, char *cdfp);
void iointerrupt(Ioproc *io);
void closeioproc(Ioproc *io);
long iocall(Ioproc *io, long (*op)(va_list *arg), ...);

DESCRIPTION
These routines provide access to I/O in slave pros. Since the I/O itself is done in a slave proc, other threads in the calling proc can run while the calling thread waits for the I/O to complete.

ioproc forks a new slave proc and returns a pointer to the Ioproc associated with it. ioproc uses mallocz and proccreate; if either fails, it calls sysfatal rather than return an error.

ioopen, ioclose, ioread, ioreadn, iowrite, and iodial execute the similarly named library or system calls (see open(2), read(2), and dial(2)) in the slave process associated with io. It is an error to execute more than one call at a time in an I/O proc.

iointerrupt interrupts the call currently executing in the I/O proc. If no call is executing, iointerrupt is a no-op.

Closeioproc terminates the I/O proc and frees the associated Ioproc.

locall is a primitive that may be used to implement more slave I/O routines. locall arranges for op to be called in io’s proc, with arg set to the variable parameter list, returning the value that op returns.

EXAMPLE
 Relay messages between two file descriptors, counting the total number of bytes seen:
   
   int tot;

   void
   relaythread(void *v)
   {
      int *fd, n;
      char buf[1024];
      Ioproc *io;

      fd = v;
      io = ioproc();
      while((n = ioread(io, fd[0], buf, sizeof buf)) > 0){
         if(iowrite(io, fd[1], buf, n) != n)
            sysfatal("iowrite: %r");
         tot += n;
      }
   closeioproc(io);

   EXAMPLE
   Relay messages between two file descriptors, counting the total number of bytes seen:
   
   int tot;

   void
   relaythread(void *v)
   {
      int *fd, n;
      char buf[1024];
      Ioproc *io;

      fd = v;
      io = ioproc();
      while((n = ioread(io, fd[0], buf, sizeof buf)) > 0){
         if(iowrite(io, fd[1], buf, n) != n)
            sysfatal("iowrite: %r");
         tot += n;
      }
   closeioproc(io);
void relay(int fd0, int fd1) {
    int fd[4];
    fd[0] = fd[3] = fd0;
    threadcreate(relaythread, fd, 8192);
    threadcreate(relaythread, fd+2, 8192);
}

If the two relaythread instances were running in different procs, the common access to tot would be unsafe.

Implement ioread:

static long _ioread(va_list *arg) {
    int fd;
    void *a;
    long n;

    fd = va_arg(*arg, int);
    a = va_arg(*arg, void*);
    n = va_arg(*arg, long);
    return read(fd, a, n);
}

long ioread(Ioproc *io, int fd, void *a, long n) {
    return iocall(io, _ioread, fd, a, n);
}

SOURCE
   /sys/src/libthread/io*.c

SEE ALSO
dial(2), open(2), read(2), thread(2)
NAME
iounit - return size of atomic I/O unit for file descriptor

SYNOPSIS
#include <u.h>
#include <libc.h>
int iounit(int fd)

DESCRIPTION
Reads and writes of files are transmitted using the 9P protocol (see intro(5)) and in general, operations involving large amounts of data must be broken into smaller pieces by the operating system. The 'I/O unit' associated with each file descriptor records the maximum size, in bytes, that may be read or written without breaking up the transfer.

The iounit routine uses the dup(3) interface to discover the I/O unit size for the file descriptor fd and returns its value. Certain file descriptors, particularly those associated with devices, may have no specific I/O unit, in which case iounit will return 0.

SOURCE
/sys/src/libc/9sys

SEE ALSO
dup(3), read(5)

DIAGNOSTICS
Returns zero if any error occurs or if the I/O unit size of the fd is unspecified or unknown.
NAME

eipfmt, parseip, parseipmask, v4parseip, v4parsecidr, parseether, myipaddr, myetheraddr, maskip, equivip4, equivip6, defmask, isv4, v4tov6, v6tov4, nhgetv, nhgetl, nhgets, hnputv, hnputl, hnputs, ptc1bsum, readipicf - Internet Protocol addressing

SYNOPSIS

#include <u.h>
#include <libc.h>
#include <ip.h>

int eipfmt(Fmt*)
vlong parseip(uchar *ipaddr, char *str)
char* v4parseip(uchar *ipaddr, char *str)
ulong v4parsecidr(uchar *addr, uchar *mask, char *str)
int parseether(uchar *eaddr, char *str)
int myetheraddr(uchar *eaddr, char *dev)
int myipaddr(uchar *ipaddr, char *net)
void maskip(uchar *from, uchar *mask, uchar *to)
int equivip4(uchar *ipaddr1, uchar *ipaddr2)
int equivip6(uchar *ipaddr1, uchar *ipaddr2)
uchar* defmask(uchar *ipaddr)
int isv4(uchar *ipaddr)
void v4tov6(uchar *ipv6, uchar *ipv4)
void v6tov4(uchar *ipv4, uchar *ipv6)
ushort nhgets(void *p)
uint nhgetl(void *p)
vulong nhgetv(void *p)
void hnputs(void *p, ushort v)
void hnputl(void *p, uint v)
void hnputv(void *p, vulong v)
ushort ptc1bsum(uchar *a, int n)
Ipifc* readipicf(char *net, Ipifc *ifc, int index)
uchar IPv4bcast[IPaddrlen];
uchar IPv4allsys[IPaddrlen];
uchar IPv4allrouter[IPaddrlen];
uchar IPallbits[IPaddrlen];
uchar IPnoaddr[IPaddrlen];
uchar v4prefix[IPaddrlen];

DESCRIPTION

These routines are used by Internet Protocol (IP) programs to manipulate IP and Ethernet addresses. Plan 9, by default, uses V6 format IP addresses. Since V4 addresses fit into the V6 space, all IP addresses can be represented. IP addresses are stored as a string of 16 unsigned chars, Ethernet addresses as 6 unsigned chars. Either V4 or V6 string representation can be used for IP addresses. For V4 addresses, the representation can be (up to) 4 decimal integers from 0 to 255 separated by periods. For V6 addresses, the representation is (up to) 8 hex integers from 0x00 to 0xFFFF separated by colons. Strings of 0 integers can be elided using two colons. For example, FFFF:1111 is equivalent to FFFF:0:0:0:0:0:0:0:1111. The string
representation for IP masks is a superset of the address representation. It includes slash notation that indicates the number of leading 1 bits in the mask. Thus, a V4 class C mask can be represented as `ffff:ffff:ffff:ffff:ffff:ffff:ff00`, `255.255.255.0`, or `/120`. The string representation of Ethernet addresses is exactly 12 hexadecimal digits.

\textit{Eipfmt} is a `print(2)` formatter for Ethernet (verb \texttt{E}) addresses, IP V6 (verb \texttt{I}) addresses, IP V4 (verb \texttt{V}) addresses, and IP V6 (verb \texttt{M}) masks.

\textit{Parseip} converts a string pointed to by \texttt{str} to a 16-byte IP address starting at \texttt{ipaddr}. As a concession to backwards compatibility, if the string is a V4 address, the return value is an unsigned long integer containing the big-endian V4 address. If not, the return value is 6. \textit{Parseipmask} converts a string pointed to by \texttt{str} to a 6-byte IP mask starting at \texttt{ipaddr}. It too returns an unsigned long big-endian V4 address or 6. Both routines return \texttt{-1} on errors.

\textit{V4parseidr} converts a string pointed to by \texttt{str} to a 4-byte V4 IP address starting at \texttt{ipaddr}.

\textit{V4pparseidr} converts a string of the form addr/mask, pointed to by \texttt{str}, to a 4-byte V4 IP address starting at \texttt{ipaddr} and a 4-byte V4 IP mask starting at \texttt{mask}.

\textit{Myipaddr} returns the first valid IP address in the IP stack rooted at \texttt{net}.

\textit{Parseether} converts a string pointed to by \texttt{str} to a 6-byte Ethernet address starting at \texttt{eaddr}. \textit{Myetheraddr} reads the Ethernet address string from file \texttt{dev/addr} and parses it into \texttt{eaddr}. Both routines return a negative number on errors.

\textit{Maskip} places the bit-wise AND of the IP addresses pointed to by its first two arguments into the buffer pointed to by the third.

\textit{Equivip} returns non-zero if the IP addresses pointed to by its two arguments are equal. \textit{Equivip4} operates on v4 addresses, \textit{equivip6} operates on v6 addresses.

\textit{Defmask} returns the standard class A, B, or C mask for \texttt{ipaddr}.

\textit{Isv4} returns non-zero if the V6 address is in the V4 space, that is, if it starts with `0:0:0:0:0:FFFF`. \textit{V4to6} converts the 4-byte V4 address, \texttt{v4ip}, to a V6 address and puts the result in \texttt{v6ip}. \textit{V6to4} converts the V6 address, \texttt{v6ip}, to a 4-byte V4 address and puts the result in \texttt{v4ip}.

\textit{Hnputs}, \texttt{hnputl} and \texttt{hnputv} are used to store 16-bit, 32-bit, and 64-bit integers, respectively, into IP big-endian form. \textit{Nhget}, \texttt{nhgetl} and \texttt{nhgetv} convert big-endian 2, 4 and 8 byte quantities into integers (or uintptrs).

\textit{Pctlbsum} returns the one’s complement checksum used in IP protocols, typically invoked as

\begin{verbatim}
hnput{hdr->cksum, ~ptclbsum(data, len) & 0xffffffff};
\end{verbatim}

A number of standard IP addresses in V6 format are also defined. They are:

\begin{verbatim}
IPV4bcast the V4 broadcast address
IPV4allsys the V4 all systems multicast address
IPV4allrouter the V4 all routers multicast address
IPAllbits the V6 all bits on address
IP0nulladdr the V6 null address, all zeros
V4prefix the V6 prefix to all embedded V4 addresses
\end{verbatim}

\textit{Readipifc} returns information about a particular interface (\texttt{index} \texttt{>= 0}) or all IP interfaces (\texttt{index} < 0) configured under a mount point \texttt{net}, default \texttt{/net}. Each interface is described by one \texttt{ipifc} structure which in turn points to a linked list of \texttt{ipifc} structures describing the addresses assigned to this interface. If the list \texttt{ifc} is supplied, that list is freed. Thus, subsequent calls can be used to free the list returned by the previous call. \texttt{ipifc} is:

\begin{verbatim}
typedef struct Ipifc
{
    Ipifc  *next; /* local addresses */
    Ipifc  *lifc; /* local addresses */
    int index; /* number of interface in ipifc dir */
    char dev[64]; /* associated physical device */
    int mtu; /* max transfer unit */
}
\end{verbatim}
uchar sendra6; /* on == send router adv */
uchar recvra6; /* on == rcv router adv */
ulong ptkin;       /* packets read */
ulong pktout;      /* packets written */
ulong errin;       /* read errors */
ulong errout;      /* write errors */
IPv6rp rp;         /* route advertisement params */
} Ipifc;

Iplifc is:
struct Iplifc
{
    Iplifc *next;
    uchar ip[IPaddrlen];
    uchar mask[IPaddrlen];
    uchar net[IPaddrlen]; /* ip & mask */
    ulong prefilt; /* preferred lifetime */
    ulong validlt; /* valid lifetime */
};

IPv6rp is:
struct IPv6rp
{
    int mflag;
    int oflag;
    int maxraint; /* max route adv interval */
    int minraint; /* min route adv interval */
    int linkmtu;
    int reachtime;
    int rxmitra;
    int ttl;
    int routerlt;
};

Dev contains the first 64 bytes of the device configured with this interface. Net is ip & mask if the network is multipoint or the remote address if the network is point to point.

SOURCE
/sys/src/libip

SEE ALSO
print(2), ip(3)
NAME
isalpharune, islowerrune, isspacechar, isitlettern, isupperrune, isdigitrune, tolowerrune, totitlettern, toupperrune – Unicode character classes and cases

SYNOPSIS
#include <u.h>
#include <libc.h>
int isalpharune(Rune c)
int islowerrune(Rune c)
int isspacechar(Rune c)
int isitlettern(Rune c)
int isupperrune(Rune c)
int isdigitrune(Rune c)
Rune tolowerrune(Rune c)
Rune totitlettern(Rune c)
Rune toupperrune(Rune c)

DESCRIPTION
These routines examine and operate on Unicode characters, in particular a subset of their properties as defined in the Unicode standard. Unicode defines some characters as alphabetic and specifies three cases: upper, lower, and title. Analogously to ctype(2) for ASCII, these routines test types and modify cases for Unicode characters. The names are self-explanatory.

The case-conversion routines return the character unchanged if it has no case.

SOURCE
/sys/src/libc/port/runetype.c

SEE ALSO
ctype(2), The Unicode Standard.
NAME
initkeyboard, ctlkeyboard, closekeyboard – keyboard control

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <thread.h>
#include <keyboard.h>

Keyboardctl *initkeyboard(char *file)
int ctlkeyboard(Keyboardctl *kc, char *msg)
void closekeyboard(Keyboardctl *kc)

DESCRIPTION
These functions access and control a keyboard interface for character-at-a-time I/O in a multi-threaded environment, usually in combination with mouse(2). They use the message-passing Channel interface in the threads library (see thread(2)); programs that wish a more event-driven, single-threaded approach should use event(2).

initkeyboard opens a connection to the keyboard and returns a Keyboardctl structure:

    typedef struct Keyboardctl Keyboardctl;
    struct Keyboardctl
    {
        Channel *c; /* chan(Rune[20]) */
        char *file;
        int consfd; /* to cons file */
        int ctlfd; /* to ctl file */
        int pid; /* of slave proc */
    };

The argument to initkeyboard is a file naming the device file from which characters may be read, typically /dev/cons. If file is nil, /dev/cons is assumed.

Once the Keyboardctl is set up a message containing a Rune will be sent on the Channel Keyboardctl.c to report each character read from the device.

ctlkeyboard is used to set the state of the interface, typically to turn raw mode on and off (see cons(3)). It writes the string msg to the control file associated with the device, which is assumed to be the regular device file name with the string ctl appended.

closekeyboard closes the file descriptors associated with the keyboard, kills the slave processes, and frees the Keyboardctl structure.

SOURCE
/sys/src/libdraw

SEE ALSO
graphics(2), draw(2), event(2), thread(2).

BUGS
Because the interface delivers complete runes, there is no way to report lesser actions such as shift keys or even individual bytes.
NAME
lock, canlock, unlock, qlock, canqlock, qunlock, rlock, canrlock, runlock, wlock, canwlock, wunlock, rsleep, rwakeup, rwakeupall, incref, decref - spin locks, queueing rendezvous locks, reader–writer locks, rendezvous points, and reference counts

SYNOPSIS
#include <u.h>
#include <libc.h>
void lock(Lock *l)
int canlock(Lock *l)
void unlock(Lock *l)
void qlock(QLock *l)
int canqlock(QLock *l)
void qunlock(QLock *l)
void rlock(RWLock *l)
int canrlock(RWLock *l)
void runlock(RWLock *l)
void wlock(RWLock *l)
int canwlock(RWLock *l)
void wunlock(RWLock *l)
typedef struct Rendez {
    QLock *l;
    ...
} Rendez;
void rsleep(Rendez *r)
int rwakeup(Rendez *r)
int rwakeupall(Rendez *r)
#include <thread.h>
typedef struct Ref {
    long ref;
} Ref;
void incref(Ref*)
long decref(Ref*)

DESCRIPTION
These routines are used to synchronize processes sharing memory.
Locks are spin locks, QLocks and RWLocks are different types of queueing rendezvous locks, and Rendezes are rendezvous points.
Locks and rendezvous points work in regular programs as well as programs that use the thread library (see thread(2)). The thread library replaces the rendezvous(2) system call with its own implementation, threadrendezvous, so that threads as well as processes may be synchronized by locking calls in threaded programs.
Used carelessly, spin locks can be expensive and can easily generate deadlocks. Their use is discouraged, especially in programs that use the thread library because they prevent context switches between threads.
Lock blocks until the lock has been obtained. Canlock is non–blocking. It tries to obtain a lock and returns a non–zero value if it was successful, 0 otherwise. Unlock releases a lock.
QLocks have the same interface but are not spin locks; instead if the lock is taken qlock will suspend execution of the calling task until it is released.
Although Locks are the more primitive lock, they have limitations; for example, they cannot synchronize between tasks in the same proc. Use QLocks instead.
RWLocks manage access to a data structure that has distinct readers and writers. Rlock grants read access; runlock releases it. Wlock grants write access; wunlock releases it. Canrlock and
canwlock are the non-blocking versions. There may be any number of simultaneous readers, but only one writer. Moreover, if write access is granted no one may have read access until write access is released.

All types of lock should be initialized to all zeros before use; this puts them in the unlocked state.

Rendezes are rendezvous points. Each Rendez r is protected by a QLock r->l, which must be held by the callers of rsleep, r wakeup, and rwakeupall. Rsleep atomically releases r->l and suspends execution of the calling task. After resuming execution, rsleep will reacquire r->l before returning. If any processes are sleeping on r, r wakeup wakes one of them. It returns 1 if a process was awakened, 0 if not. Rwakeupall wakes all processes sleeping on r, returning the number of processes awakened. Rwakeup and rwakeupall do not release r->l and do not suspend execution of the current task.

Before use, Rendezes should be initialized to all zeros except for r->l pointer, which should point at the QLock that will guard r. It is important that this QLock is the same one that protects the rendezvous condition; see the example.

A Ref contains a long that can be incremented and decremented atomically: Incref increments the Ref in one atomic operation. Decref atomically decrements the Ref and returns zero if the resulting value is zero, non-zero otherwise.

EXAMPLE

Implement a buffered single-element channel using rsleep and r wakeup:

```c
typedef struct Chan {
  QLock l;
  Rendez full, empty;
  int val, haveval;
} Chan;
Chan* mkchan(void) {
  Chan *c;
  c = mallocz(sizeof *c, 1);
  c->full.l = &c->l;
  c->empty.l = &c->l;
  return c;
}
void send(Chan *c, int val) {
  qlock(&c->l);
  while(c->haveval)
    rsleep(&c->full);
  c->haveval = 1;
  c->val = val;
  rwakeup(&c->empty); /* no longer empty */
  qunlock(&c->l);
}
int recv(Chan *c) {
  int v;
  qlock(&c->l);
  while(!c->haveval)
    rsleep(&c->empty);
  c->haveval = 0;
  v = c->val;
  return v;
}
```
rwakeup(&c->full); /* no longer full */
qunlock(&c->l);
return v;
}

Note that the QLock protecting the Chan is the same QLock used for the Rendez; this ensures that wakeups are not missed.

SOURCE
/sys/src/libc/port/lock.c
/sys/src/libc/9sys/qlock.c
/sys/src/libthread/ref.c

SEE ALSO
rfork in fork(2)

BUGS
Locks are not strictly spin locks. After each unsuccessful attempt, lock calls sleep(0) to yield the CPU; this handles the common case where some other process holds the lock. After a thousand unsuccessful attempts, lock sleeps for 100ms between attempts. After another thousand unsuccessful attempts, lock sleeps for a full second between attempts. Locks are not intended to be held for long periods of time. The 100ms and full second sleeps are only heuristics to avoid tying up the CPU when a process deadlocks. As discussed above, if a lock is to be held for much more than a few instructions, the queueing lock types should be almost always be used.

It is an error for a program to fork when it holds a lock in shared memory, since this will result in two processes holding the same lock at the same time, which should not happen.
NAME

- crackhdr
- machbytype
- machbyname
- newmap
- setmap
- findseg
- unusemap
- loadmap
- attachproc
- get1
- get2
- get4
- get8
- put1
- put2
- put4
- put8
- beswab
- beswal
- beswav
- leswab
- leswal
- leswav

Machine-independent access to executable files

SYNOPSIS

```c
#include <u.h>
#include <libc.h>
#include <bio.h>
#include <mach.h>

int crackhdr(int fd, Fhdr *fp)
void machbytype(int type)
int machbyname(char *name)
Map *newmap(Map *map, int n)
int setmap(Map *map, int fd, ulong base, ulong end,
          ulong foffset, char *name)
int findseg(Map *map, char *name)
void unusemap(Map *map, int seg)
Map *loadmap(Map *map, int fd, Fhdr *fp)
Map *attachproc(int pid, int kflag, int corefd, Fhdr *fp)
int get1(Map *map, ulong addr, uchar *buf, int n)
int get2(Map *map, ulong addr, ushort *val)
int get4(Map *map, ulong addr, long *val)
int get8(Map *map, ulong addr, vlong *val)
int put1(Map *map, ulong addr, uchar *buf, int n)
int put2(Map *map, ulong addr, ushort val)
int put4(Map *map, ulong addr, long val)
int put8(Map *map, ulong addr, vlong val)
ushort beswab(ushort val)
long beswal(long val)
long beswav(vlong val)
ushort leswab(ushort val)
long leswal(long val)
long leswav(vlong val)
extern Mach mach;
extern Machdata machdata;
```

DESCRIPTION

These functions provide a processor-independent interface for accessing the executable files or executing images of all architectures. Related library functions described in `symbol(2)` and `object(2)` provide similar access to symbol tables and object files.

An executable is a file containing an executable program or the text file of the `/proc` file system associated with an executing process as described in `proc(3)`. After opening an executable, an application invokes a library function which parses the file header, determines the target architecture and initializes data structures with parameters and pointers to functions appropriate for that architecture. Next, the application invokes functions to construct one or more maps, data structures that translate references in the address space of the executable to offsets in the file. Each map comprises one or more segments, each associating a non-overlapping range of memory
addresses with a logical section of the executable. Other library functions then use a map and the architecture–specific data structures to provide a generic interface to the processor–dependent data.

Crankhdr interprets the header of the executable associated with the open file descriptor fd. It loads the data structure fp with a machine–independent description of the header information and points global variable mach to the Mach data structure containing processor–dependent parameters of the target architecture.

Machbytype selects architecture–specific data structures and parameter values based on the code stored in the field named type in the Fhdr data structure. Machbyname performs the same selection based on the name of a processor class; see 2c(1) for a list of valid names. Both functions point global variables mach and machdata to the Mach and Machdata data structures appropriate for the target architecture and load global variable asstype with the proper disassembler type code.

Newmap creates an empty map with n segments. If map is zero, the new map is dynamically allocated, otherwise it is assumed to point to an existing dynamically allocated map whose size is adjusted, as necessary. A zero return value indicates an allocation error.

Setmap loads the first unused segment in map with the segment mapping parameters. Fd is an open file descriptor associated with an executable. Base and end contain the lowest and highest virtual addresses mapped by the segment. Offset is the offset to the start of the segment in the file. Name is a name to be attached to the segment.

Findseg returns the index of the the segment named name in map. A return of –1 indicates that no segment matches name.

Unusemap marks segment number seg in map map unused. Other segments in the map remain unaffected.

Loadmap initializes a default map containing segments named ‘text’ and ‘data’ that map the instruction and data segments of the executable described in the Fhdr structure pointed to by fp. Usually that structure was loaded by crankhdr and can be passed to this function without modification. If map is non–zero, that map, which must have been dynamically allocated, is resized to contain two segments; otherwise a new map is allocated. This function returns zero if allocation fails. Loadmap is usually used to build a map for accessing a static executable, for example, an executable program file.

Attachproc constructs a map for accessing a running process. It returns the address of a Map containing segments mapping the address space of the running process whose process ID is p.id. If kflag is non–zero, the process is assumed to be a kernel process. CoreEd is an file descriptor opened to /proc/pid/mem. Fp points to the Fhdr structure describing the header of the executable. For most architectures the resulting Map contains four segments named ‘text’, ‘data’, ‘regs’ and ‘fregs’. The latter two provide access to the general and floating point registers, respectively. If the executable is a kernel process (indicated by a non–zero kflag argument), the data segment extends to the maximum supported address, currently 0xffffffff, and the register sets are read–only. In user–level programs, the data segment extends to the top of the stack or 0x7fffffff if the stack top cannot be found, and the register sets are readable and writable. Attachproc returns zero if it is unable to build the map for the specified process.

Get1, get2, get4, and get8 retrieve the data stored at address addr in the executable associated with map. Get1 retrieves n bytes of data beginning at addr into buf. Get2, get4 and get8 retrieve 16–bit, 32–bit and 64–bit values respectively, into the location pointed to by val. The value is byte–swapped if the source byte order differs from that of the current architecture. This implies that the value returned by get2, get4, and get8 may not be the same as the byte sequences returned by get1 when n is two, four or eight; the former may be byte–swapped, the latter reflects the byte order of the target architecture. If the file descriptor associated with the applicable segment in map is negative, the address itself is placed in the return location. These functions return the number of bytes read or a –1 when there is an error.

Put1, put2, put4, and put8 write to the executable associated with map. The address is translated using the map parameters and multi–byte quantities are byte–swapped, if necessary, before they are written. Put1 transfers n bytes stored at buf; put2, put4, and put8 write the 16–bit, 32–bit or 64–bit quantity contained in val, respectively. The number of bytes transferred is returned. A –1
return value indicates an error.

Beswab, beswal, and beswav return the ushort, long, and vlong big-endian representation of
val, respectively. Leswab, leswal, and leswav return the little-endian representation of the
ushort, long, and vlong contained in val.

SOURCE
/sys/src/libmach

SEE ALSO
2c(1), symbol(2), object(2), errstr(2), proc(3), a.out(6)

DIAGNOSTICS
These routines set errstr.
NAME
malloc, mallocalign, mallocz, free, realloc, calloc, msize, setmallocctag, setreallocctag, getmallocctag,
getreallocctag, malloctopoolblock – memory allocator

SYNOPSIS
#include <u.h>
#include <libc.h>
void* malloc(ulong size)
void* mallocalign(ulong size, ulong align, long offset, ulong span)
void* mallocz(ulong size, int clr)
void free(void *ptr)
void* realloc(void *ptr, ulong size)
void* calloc(ulong nelem, ulong elsize)
ulong msize(void *ptr)
void setmallocctag(void *ptr, ulong tag)
ulong getmallocctag(void *ptr)
void setreallocctag(void *ptr, ulong tag)
ulong getreallocctag(void *ptr)
void* malloctopoolblock(void*)

DESCRIPTION
Malloc and free provide a simple memory allocation package.Malloc returns a pointer to a new
block of at least size bytes. The block is suitably aligned for storage of any type of object. No two
active pointers from malloc will have the same value. The call malloc(0) returns a valid pointer
rather than null.

The argument to free is a pointer to a block previously allocated by malloc; this space is made
available for further allocation. It is legal to free a null pointer; the effect is a no-op. The contents
of the space returned by malloc are undefined. Mallocz behaves as malloc, except that if clr is
non-zero, the memory returned will be zeroed.

Mallocalign allocates a block of at least n bytes of memory respecting alignment constraints. If
align is non-zero, the returned pointer is aligned to be equal to offset modulo align. If span is
non-zero, the n byte block allocated will not span a span-byte boundary.

Realloc changes the size of the block pointed to by ptr to size bytes and returns a pointer to the
(possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.
Realloc takes on special meanings when one or both arguments are zero:

realloc(0, size)
    means malloc(size); returns a pointer to the newly–allocated memory

realloc(ptr, 0)
    means free(ptr); returns null

realloc(0, 0)
    no-op; returns null

Calloc allocates space for an array of nelem elements of size elsize. The space is initialized to
zeros. Free frees such a block.

When a block is allocated, sometimes there is some extra unused space at the end. Msize grows
the block to encompass this unused space and returns the new number of bytes that may be used.

The memory allocator maintains two word–sized fields associated with each block, the “malloc
tag” and the “realloc tag”. By convention, the malloc tag is the PC that allocated the block, and
the realloc tag the PC that last reallocated the block. These may be set or examined with
setmallocctag, getmallocctag, setreallocctag, and getreallocctag. When allocating blocks directly with
malloc and realloc, these tags will be set properly. If a custom allocator wrapper is used, the allo-
cator wrapper can set the tags itself (usually by passing the result of getcallerpc(2) to
setmalloctag) to provide more useful information about the source of allocation.

Mallocopoolblock takes the address of a block returned by malloc and returns the address of the corresponding block allocated by the pool(2) routines.

SOURCE
/sys/src/libc/port/malloc.c

SEE ALSO
leak(1), trump (in acid(1)), brk(2), getcallerpc(2), pool(2)

DIAGNOSTICS
Malloc, realloc and calloc return 0 if there is no available memory. Errstr is likely to be set. If the allocated blocks have no malloc or realloc tags, getmalloctag and getrealloctag return ~0.

After including pool.h, the call poolcheck(mainmem) can be used to scan the storage arena for inconsistencies such as data written beyond the bounds of allocated blocks. It is often useful to combine this with with setting

mainmem->flags |= POOL_NOREUSE;

at the beginning of your program. This will cause malloc not to reallocate blocks even once they are freed; poolcheck(mainmem) will then detect writes to freed blocks.

The trump library for acid can be used to obtain traces of malloc execution; see acid(1).

BUGS
The different specification of calloc is bizarre.

User errors can corrupt the storage arena. The most common gaffes are (1) freeing an already freed block, (2) storing beyond the bounds of an allocated block, and (3) freeing data that was not obtained from the allocator. When malloc and free detect such corruption, they abort.
NAME
ident, matmul, matmulr, determinant, adjoint, invertmat, xformpoint, xformpointd, xformplane,
pushmat, popmat, rot, qrot, scale, move, xform, ixform, persp, look, viewport – Geometric transformations

SYNOPSIS
#include <draw.h>
#include <geometry.h>
void ident(Matrix m)
void matmul(Matrix a, Matrix b)
void matmulr(Matrix a, Matrix b)
double determinant(Matrix m)
void adjoint(Matrix m, Matrix madj)
double invertmat(Matrix m, Matrix inv)
Point3 xformpoint(Point3 p, Space *to, Space *from)
Point3 xformpointd(Point3 p, Space *to, Space *from)
Point3 xformplane(Point3 p, Space *to, Space *from)
Space *pushmat(Space *t)
Space *popmat(Space *t)
void rot(Space *t, double theta, int axis)
void qrot(Space *t, Quaternion q)
void scale(Space *t, double x, double y, double z)
void move(Space *t, double x, double y, double z)
void xform(Space *t, Matrix m)
void ixform(Space *t, Matrix m, Matrix inv)
int persp(Space *t, double fov, double n, double f)
void look(Space *t, Point3 eye, Point3 look, Point3 up)
void viewport(Space *t, Rectangle r, double aspect)

DESCRIPTION
These routines manipulate 3-space affine and projective transformations, represented as 4×4 matrices, thus:

typedef double Matrix[4][4];

Ident stores an identity matrix in its argument. Matmul stores a×b in a. Matmulr stores b×a in b. Determinant returns the determinant of matrix m. Adjoint stores the adjoint (matrix of cofactors) of m in madj. Invertmat stores the inverse of matrix m in minv, returning m’s determinant. Should m be singular (determinant zero), invertmat stores its adjoint in minv.

The rest of the routines described here manipulate Spaces and transform Point3s. A Point3 is a point in three-space, represented by its homogeneous coordinates:

typedef struct Point3 Point3;
struct Point3{
    double x, y, z, w;
};

The homogeneous coordinates (x, y, z, w) represent the Euclidean point (x/w, y/w, z/w) if w≠0, and a “point at infinity” if w=0.

A Space is just a data structure describing a coordinate system:

typedef struct Space Space;
struct Space{
Matrix t;
Matrix tinv;
Space *next;
}

It contains a pair of transformation matrices and a pointer to the Space's parent. The matrices transform points to and from the "root coordinate system," which is represented by a null Space pointer.

Pushmat creates a new Space. Its argument is a pointer to the parent space. Its result is a newly allocated copy of the parent, but with its next pointer pointing at the parent. Popmat discards the Space that is its argument, returning a pointer to the stack. Namely, these two functions define a stack of transformations, but pushmat can be called multiple times on the same Space multiple times, creating a transformation tree.

Xformpoint and Xformpointd both transform points from the Space pointed to by from to the space pointed to by to. Either pointer may be null, indicating the root coordinate system. The difference between the two functions is that xformpointd divides x, y, z, and w by w, if w≠0, making (x, y, z) the Euclidean coordinates of the point.

Xformplane transforms planes or normal vectors. A plane is specified by the coefficients (a, b, c, d) of its implicit equation ax+by+cz+d=0. Since this representation is dual to the homogeneous representation of points, libgeometry represents planes by Point3 structures, with (a, b, c, d) stored in (x, y, z, w).

The remaining functions transform the coordinate system represented by a Space. Their Space * argument must be non-null — you can't modify the root Space. Rot rotates by angle theta (in radians) about the given axis, which must be one of XAXIS, YAXIS or ZAXIS. Qrot transforms by a rotation about an arbitrary axis, specified by Quaternion q.

Scale scales the coordinate system by the given scale factors in the directions of the three axes. Move translates by the given displacement in the three axial directions.

Xform transforms the coordinate system by the given Matrix. If the matrix's inverse is known a priori, calling xform will save the work of recomputing it.

Persp does a perspective transformation. The transformation maps the frustrum with apex at the origin, central axis down the positive y axis, and apex angle fov and clipping planes y=n and y=f into the double-unit cube. The plane y=n maps to y′=-1, y=f maps to y′=1.

Look does a view-pointing transformation. The eye point is moved to the origin. The line through the eye and look points is aligned with the y axis, and the plane containing the eye, look and up points is rotated into the x-y plane.

Viewport maps the unit-cube window into the given screen viewport. The viewport rectangle r has r.min at the top left-hand corner, and r.max just outside the lower right-hand corner. Argument aspect is the aspect ratio (dx/dy) of the viewport's pixels (not of the whole viewport). The whole window is transformed to fit centered inside the viewport with equal slop on either top and bottom or left and right, depending on the viewport's aspect ratio. The window is viewed down the y axis, with x to the left and z up. The viewport has x increasing to the right and y increasing down. The window's y coordinates are mapped, unchanged, into the viewport's z coordinates.

SOURCE
/sys/src/libgeometry/matrix.c

SEE ALSO
arith3(2)
NAME
Memimage, Memdata, Memdrawparam, memimageinit, wordaddr, byteaddr, memimagemove,
allocmemimage, allocmemimaged, readmemimage, creadmemimage, writememimage, freememimag-
eme, memsetchan, loadmemimage, cloadmemimage, unloadmemimage, memfillcolor, memarc,
mempoly, memellipses, memfillpoly, memimageline, memimagedraw, drawclip, memlinebbox,
memlineendsize, allocmemsubfont, openmemsubfont, freememsubfont, memsubfontwidth, get-
memdefont, memimagestring, iprint, hwdraw - drawing routines for memory-resident images

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <memdraw.h>
typedef struct Memdata
{
    ulong *base; /* allocated data pointer */
    uchar *bdata; /* first byte of actual data; word-aligned */
    int ref; /* number of Memimages using this data */
    void* imref; /* last image that pointed at this */
    int allocd; /* is this malloc'd? */
} Memdata;
enum {
    Frepl = 1<<0, /* is replicated */
    Fsimple = 1<<1, /* is 1x1 */
    Fgrey = 1<<2, /* is grey */
    Falpha = 1<<3, /* has explicit alpha */
    Fcmap = 1<<4, /* has cmap channel */
    Fbytes = 1<<5, /* has only 8-bit channels */
};
typedef struct Memimage
{
    Rectangle r; /* rectangle in data area, local coords */
    Rectangle clipr; /* clipping region */
    int depth; /* number of bits of storage per pixel */
    int nchan; /* number of channels */
    ulong chan; /* channel descriptions */
    Memdata *data; /* pointer to data */
    int zero; /* data->bdata+zero==&byte containing (0,0) */
    ulong width; /* width in words of a single scan line */
    Memlayer *layer; /* nil if not a layer*/
    ulong flags;
} Memimage;
typedef struct Memdrawparam
{
    Memimage *dst;
    Rectangle r;
    Memimage *src;
    Rectangle sr;
    Memimage *mask;
    Rectangle mr;
} Memdrawparam;
int drawdebug;

void memimageinit(void)
ulong* wordaddr(Memimage *i, Point p)
uchar* byteaddr(Memimage *i, Point p)
void memimagemove(void *from, void *to)

Memimage* allocmemimage(Rectangle r, ulong chan)
Memimage* allocmemimaged(Rectangle r, ulong chan, Memdata *data)
Memimage* readmemimage(int fd)
Memimage* creadmemimage(int fd)
int writememimage(int fd, Memimage *i)
void freememimage(Memimage *i)
int memsetchan(Memimage*, ulong)
int loadmemimage(Memimage *i, Rectangle r, uchar *buf, int nbuf)
int cloadmemimage(Memimage *i, Rectangle r, uchar *buf, int nbuf)
int unloadmemimage(Memimage *i, Rectangle r, uchar *buf, int nbuf)
void memfillcolor(Memimage *i, ulong color)
void memarc(Memimage *dst, Point c, int a, int b, int thick,
            Memimage *src, Point sp, Drawop op)
void mempolly(Memimage *dst, Point p, Point sp, int alpha, int phi, Drawop op)
void memellipse(Memimage *dst, Point c, int a, int b,
                Memimage *src, Point sp, Drawop op)
void memfillpoly(Memimage *dst, Point *p, int np, int wind,
                Memimage *src, Point sp, Drawop op)
void memimageline(Memimage *dst, Point p0, Point p1, int end0,
                 Memimage *src, Point sp, Drawop op)
int drawclip(Memimage *dst, Rectangle *dr, Memimage *src,
            Point *sp, Memimage *mask, Point *mp,
            Rectangle *sr, Rectangle *mr)

Rectangle memlinebbox(Point p0, Point p1, int end0, int end1,
                      int radius)
int memlineendsize(int end)
Memsubfont* allocmemsubfont(char *name, int n, int height,
                             int ascent, Fontchar *info, Memimage *i)
Memsubfont* openmemsubfont(char *name)
void freememsubfont(Memsubfont *f)
Point memsubfontwidth(Memsubfont *f, char *s)
Memsubfont* getmendefont(void)
Point memimagestring(Memimage *dst, Point p, Memimage *color,
                    Point cp, Memsubfont *f, char *cs)

int iprint(char *fmt, ...)
int hwdraw(Memdrawparam *param)

DESCRIPTION
The Memimage type defines memory–resident rectangular pictures and the methods to draw upon
them; Memimages differ from Images (see draw(2)) in that they are manipulated directly in user
memory rather than by RPCs to the /dev/draw hierarchy. The memdraw library is the basis for
the kernel draw(3) driver and also used by a number of programs that must manipulate images
without a display.

The r, clipr, depth, nchan, and chan structure elements are identical to the ones of the
same name in the Image structure.
The flags element of the Memimage structure holds a number of bits of information about the image. In particular, it subsumes the purpose of the repl element of Image structures.

Memimageinit initializes various static data that the library depends on, as well as the replicated solid color images memopaque, memtransparent, memblack, and memwhite. It should be called before referring to any of these images and before calling any of the other library functions.

Each Memimage points at a Memdata structure that in turn points at the actual pixel data for the image. This allows multiple images to be associated with the same Memdata. The first word of the data pointed at by the base element of Memdata points back at the Memdata structure, so that the memory allocator (see pool(2)) can compact image memory using memimagemove.

Because images can have different coordinate systems, the zero element of the Memimage structure contains the offset that must be added to the bdata element of the corresponding Memdata structure in order to yield a pointer to the data for the pixel (0,0). Adding width machine words to this pointer moves it down one scan line. The depth element can be used to determine how to move the pointer horizontally. Note that this method works even for images whose rectangles do not include the origin, although one should only dereference pointers corresponding to pixels within the image rectangle. Wordaddr and byteaddr perform these calculations, returning pointers to the word and byte, respectively, that begin the data of a given pixel.

Allocmemimage allocates images with a given rectangle and channel descriptor (see strtochan in graphics(2)), creating a fresh Memdata structure and associated storage. Allocmemimage is similar but uses the supplied Memdata structure rather than a new one. The readmemimage function reads an uncompressed bitmap from the given file descriptor, while creadmemimage reads a compressed bitmap. Writememimage writes a compressed representation of i to file descriptor fd. For more on bitmap formats, see image(6). Freememimage frees images returned by any of these routines. The Memimage structure contains some tables that are used to store precomputed values depending on the channel descriptor. Memsetchan updates the chan element of the structure as well as these tables, returning −1 if passed a bad channel descriptor.

Loadmemimage and cloudmemimage replace the pixel data for a given rectangle of an image with the given buffer of uncompressed or compressed data, respectively. When calling cloudmemimage, the buffer must contain an integral number of compressed chunks of data that exactly cover the rectangle. Unloadmemimage retrieves the uncompressed pixel data for a given rectangle of an image. All three return the number of bytes consumed on success, and −1 in case of an error.

Memfillcolor fills an image with the given color, a 32–bit number as described in color(2).

Memarc, mempoly, memellipse, memfillpoly, memmageline, and memimagedraw are identical to the arc, poly, ellipse, filpoly, line, and gendraw, routines described in draw(2), except that they operate on Memimages rather than Images. Similarly, allocmemsubfont, openmemsubfont, freememsubfont, memsubfontwidth, getmemdefont, and memimagestring are the Memimage analogues of allocsubfont, openfont, freesubfont, strsubfontwidth, getdefont, and string (see subfont(2) and graphics(2)), except that they operate only on MemSubfonts rather than Fonts.

Drawclip takes the images involved in a draw operation, together with the destination rectangle dxr and source and mask alignment points sp and mp, and clips them according to the clipping rectangles of the images involved. It also fills in the rectangles sxr and mxr with rectangles congruent to the returned destination rectangle but translated so the upper left corners are the returned sp and mp. Drawclip returns zero when the clipped rectangle is empty. Memlinebbox returns a conservative bounding box containing a line between two points with given end styles and radius. Memlineendsize calculates the extra length added to a line by attaching an end of a given style.

The hwdraw and iprint functions are no–op stubs that may be overridden by clients of the library. Hwdraw is called at each call to memimagedraw with the current request’s parameters. If it can satisfy the request, it should do so and return 1. If it cannot satisfy the request, it should return 0. This allows (for instance) the kernel to take advantage of hardware acceleration. Iprint should format and print its arguments; it is given much debugging output when the global integer variable drawdebug is non–zero. In the kernel, iprint prints to a serial line rather than the screen, for obvious reasons.
SOURCE
/sys/src/libmemdraw

SEE ALSO
addpt(2), color(2), draw(2), graphics(2), memlayer(2), stringsize(2), subfont(2), color(6), utf(6)

BUGS
Memimagestring is unusual in using a subfont rather than a font, and in having no parameter to align the source.
NAME
memdraw, memlalloc, memldelete, memlexpose, memlfree, memlhide, memline, memlnorefresh,
memload, memunload, memlorigin, memlsetrefresh, memltofront, memltofrontn, memltorear,
memltorearn – windows of memory-resident images

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <memdraw.h>
#include <memlayer.h>

typedef struct Memscreen Memscreen;
typedef struct Memlayer Memlayer;
typedef void (*Refreshfn)(Memimage*, Rectangle, void*);

struct Memscreen
{
    Memimage *frontmost; /* frontmost layer on screen */
    Memimage *rearmost; /* rearmost layer on screen */
    Memimage *image; /* upon which all layers are drawn */
    Memimage *fill; /* if non-zero, picture to use when repainting */
};

struct Memlayer
{
    Rectangle screenr; /* true position of layer on screen */
    Point delta; /* add delta to go from image coords to screen */
    Memscreen *screen; /* screen this layer belongs to */
    Memimage *front; /* window in front of this one */
    Memimage *rear; /* window behind this one*/
    int clear; /* layer is fully visible */
    Memimage *save; /* save area for obscured parts */
    Refreshfn refreshfn; /* fn to refresh obscured parts if save=nil */
    void *refreshptr; /* argument to refreshfn */
};
Memimage* memlalloc(Memscreen *s, Rectangle r, Refreshfn fn, void *arg, ulong col)
void memlnorefresh(Memimage *i, Rectangle r, void *arg)
int memlsetrefresh(Memimage *i, Refreshfn fn, void *arg)
int memldelete(Memimage *i)
int memlfree(Memimage *i)
int memlexpose(Memimage *i)
int memlhide(Memimage *i, Rectangle r)
void memltofront(Memimage *i)
void memltofrontn(Memimage **ia, int n)
void memltorear(Memimage *i)
void memltorearn(Memimage **ia , int n)
int memlorigin(Memimage *i, Point log, Point phys)
void memdraw(Image *dst, Rectangle r, Image *src, Point sp, Image *mask, Point mp, Drawop op)
int memload(Memimage *i, Rectangle r, uchar *buf, int n, int iscompressed)
int memunload(Memimage *i, Rectangle r, uchar *buf, int n)
DESCRIPTION

These functions build upon the memdraw(2) interface to maintain overlapping graphical windows on in-memory images. They are used by the kernel to implement the windows interface presented by draw(3) and window(2) and probably have little use outside of the kernel.

The basic function is to extend the definition of a Memimage (see memdraw(2)) to include overlapping windows defined by the MemLayer type. The first fields of the Memlayer structure are identical to those in Memimage, permitting a function that expects a Memimage to be passed a MemLayer, and vice versa. Both structures have a save field, which is nil in a Memimage and points to ‘backing store’ in a Memlayer. The layer routines accept Memimages or Memlayers; if the image is a Memimage the underlying Memimage routine is called; otherwise the layer routines recursively subdivide the geometry, reducing the operation into a smaller component that ultimately can be performed on a Memimage, either the display on which the window appears, or the backing store.

Memlayers are associated with a Memscreen that holds the data structures to maintain the windows and connects them to the associated image. The fill color is used to paint the background when a window is deleted. There is no function to establish a Memscreen; to create one, allocate the memory, zero frontmost and rearmost, set fill to a valid fill color or image, and set image to the Memimage (or MemLayer) on which the windows will be displayed.

Memmalloc allocates a Memlayer of size r on Memscreen s. If col is not DNoFill, the new window will be initialized by painting it that color.

The refresh function fn and associated argument arg will be called by routines in the library to restore portions of the window uncovered due to another window being deleted or this window being pushed to the front of the stack. The function, when called, receives a pointer to the image (window) being refreshed, the rectangle that has been uncovered, and the arg recorded when the window was created. A couple of predefined functions provide built-in management methods: memnorefresh does no backup at all, useful for making efficient temporary windows; while a nil function specifies that the backing store (Memlayer.save) will be used to keep the obscured data. Other functions may be provided by the client. Memsetrefresh allows one to change the function associated with the window.

MemDelete deletes the window i, restoring the underlying display. MemFree frees the data structures without unlinking the window from the associated Memscreen or doing any graphics.

MemExpose restores rectangle r within the window, using the backing store or appropriate refresh method. Memhide goes the other way, backing up r so that that portion of the screen may be modified without losing the data in this window.

MemToFront pulls i to the front of the stack of windows, making it fully visible. MemToFrontr pulls the n windows in the array ia to the front as a group, leaving their internal order unaffected. MemToBack and memtorear push the windows to the rear.

Memloorigin changes the coordinate systems associated with the window i. The points log and phys represent the upper left corner (min) of the window’s internal coordinate system and its physical location on the screen. Changing log changes the interpretation of coordinates within the window; for example, setting it to (0, 0) makes the upper left corner of the window appear to be the origin of the coordinate system, regardless of its position on the screen. Changing phys changes the physical location of the window on the screen. When a window is created, its logical and physical coordinates are the same, so

\[ \text{memloorigin}(i, i\rightarrow\text{r.min}, i\rightarrow\text{r.min}) \]

would be a no-op.

Memdraw and memline are implemented in the layer library but provide the main entry points for drawing on memory-resident windows. They have the signatures of memimagedraw and memimageline (see memdraw(2)) but accept MemLayer or MemImage arguments both.

Memload and memunload are similarly layer-savvy versions of loadmemimage and unloadmemimage. The iscompressed flag to memload specifies whether the \( n \) bytes of data in buf are in compressed image format (see image(6)).

SOURCE

/sys/src/libmemlayer
SEE ALSO

graphics(2), memdraw(2), stringsize(2), window(2), draw(3)
NAME
memccpy, memchr, memcmp, memcpy, memmove, memset - memory operations

SYNOPSIS
#include <u.h>
#include <libc.h>
void* memccpy(void *s1, void *s2, int c, ulong n)
void* memchr(void *s, int c, ulong n)
int memcmp(void *s1, void *s2, ulong n)
void* memcpy(void *s1, void *s2, ulong n)
void* memmove(void *s1, void *s2, ulong n)
void* memset(void *s, int c, ulong n)

DESCRIPTION
These functions operate efficiently on memory areas (arrays of bytes bounded by a count, not
terminated by a zero byte). They do not check for the overflow of any receiving memory area.
Memccpy copies bytes from memory area s2 into s1, stopping after the first occurrence of byte c
has been copied, or after n bytes have been copied, whichever comes first. It returns a pointer to
the byte after the copy of c in s1, or zero if c was not found in the first n bytes of s2.
Memchr returns a pointer to the first occurrence of byte c in the first n bytes of memory area s, or
zero if c does not occur.
Memcmp compares its arguments, looking at the first n bytes only, and returns an integer less
than, equal to, or greater than 0, according as s1 is lexicographically less than, equal to, or greater
than s2. The comparison is bytwise unsigned.
Memcpy copies n bytes from memory area s2 to s1. It returns s1.
Memmove works like memcpy, except that it is guaranteed to work if s1 and s2 overlap.
Memset sets the first n bytes in memory area s to the value of byte c. It returns s.

SOURCE
All these routines have portable C implementations in /sys/src/libc/port. Most also have
machine-dependent assembly language implementations in /sys/src/libc/$objtype.

SEE ALSO
strcat(2)

BUGS
ANSI C does not require memcpy to handle overlapping source and destination; on Plan 9, it does,
so memmove and memcpy behave identically.
If memcpy and memmove are handed a negative count, they abort.
NAME
  mktemp - make a unique file name

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  char* mktemp(char *template)

DESCRIPTION
  Mktemp replaces template by a unique file name, and returns the address of the template. The
  template should look like a file name with eleven trailing Xs. The Xs are replaced by a letter fol-
  lowed by the current process id. Letters from a to z are tried until a name that can be accessed
  (see access(2)) is generated. If no such name can be generated, mktemp returns "/".

SOURCE
  /sys/src/libc/port/mktemp.c

SEE ALSO
  getpid(2), access(2)
NAME
initmouse, readmouse, closemouse, moveto, getrect, drawgetrect, menuhit, setcursor - mouse control

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <thread.h>
#include <mouse.h>
#include <cursor.h>
Mousectl *initmouse(char *file, Image *i)
int readmouse(Mousectl *mc)
int atomouse();
void closemouse(Mousectl *mc)
void moveto(Mousectl *mc, Point pt)
void setcursor(Mousectl *mc, Cursor *c)
Rectangle getrect(int but, Mousectl *mc)
void drawgetrect(Rectangle r, int up)
int menuhit(int but, Mousectl *mc, Menu *menu, Screen *scr)

DESCRIPTION
These functions access and control a mouse in a multi-threaded environment. They use the message-passing Channel interface in the threads library (see thread(2)); programs that wish a more event-driven, single-threaded approach should use event(2).

The state of the mouse is recorded in a structure, Mouse, defined in <mouse.h>:

typedef struct Mouse Mouse;
struct Mouse
{
    int    buttons; /* bit array: LMR=124 */
    Point  xy;
    ulong  msec;
};

The Point xy records the position of the cursor, buttons the state of the buttons (three bits representing, from bit 0 up, the buttons from left to right, 0 if the button is released, 1 if it is pressed), and msec, a millisecond time stamp.

The routine initmouse returns a structure through which one may access the mouse:

typedef struct Mossectl Mousectl;
struct Mousectl
{
    Mouse;
    Channel  *c; /* chan(Mouse)[16] */
    Channel  *resizec; /* chan(int)[2] */
    char     *file;
    int       mfd; /* to mouse file */
    int       cfd; /* to cursor file */
    int       pid; /* of slave proc */
    Image*    image; /* of associated window/display */
};

The arguments to initmouse are a file naming the device file connected to the mouse and an Image (see draw(2)) on which the mouse will be visible. Typically the file is nil, which requests the default /dev/mouse; and the image is the window in which the program is running, held in the
variable screen after a call to initdraw.

Once the Mousectl is set up, mouse motion will be reported by messages of type Mouse sent on the Channel Mousectl.c. Typically, a message will be sent every time a read of /dev/mouse succeeds, which is every time the state of the mouse changes.

When the window is resized, a message is sent on Mousectl.resizec. The actual value sent may be discarded; the receipt of the message tells the program that it should call getwindow (see graphics(2)) to reconnect to the window.

Readmouse updates the Mouse structure held in the Mousectl, blocking if the state has not changed since the last readmouse or message sent on the channel. It calls flushimage (see graphics(2)) before blocking, so any buffered graphics requests are displayed.

Closemouse closes the file descriptors associated with the mouse, kills the slave processes, and frees the Mousectl structure.

Moveto moves the mouse cursor on the display to the position specified by pt.

Setcursor sets the image of the cursor to that specified by c. If c is nil, the cursor is set to the default. The format of the cursor data is spelled out in <cursor.h> and described in graphics(2).

Getrect returns the dimensions of a rectangle swept by the user, using the mouse, in the manner rio(1) or sam(1) uses to create a new window. The but argument specifies which button the user must press to sweep the window; any other button press cancels the action. The returned rectangle is all zeros if the user cancels.

Getrect uses successive calls to drawgetrect to maintain the red rectangle showing the sweep-in-progress. The rectangle to be drawn is specified by rc and the up parameter says whether to draw (1) or erase (0) the rectangle.

Menuhit provides a simple menu mechanism. It uses a Menu structure defined in <mouse.h>:

```c
typedef struct Menu Menu;
struct Menu
{
    char  **item;
    char  *(*gen)(int);
    int   lasthit;
};
```

Menuhit behaves the same as its namesake emenuhit described in event(2), with two exceptions. First, it uses a Mousectl to access the mouse rather than using the event interface; and second, it creates the menu as a true window on the Screen scr (see window(2)), permitting the menu to be displayed in parallel with other activities on the display. If scr is null, menuhit behaves like emenuhit, creating backing store for the menu, writing the menu directly on the display, and restoring the display when the menu is removed.

SOURCE
/sys/src/libdraw

SEE ALSO
graphics(2), draw(2), event(2), keyboard(2), thread(2).
NAME
mpsetminbits, mpnew, mpfree, mpbits, mpnorm, mpnorm, mpcopy, mpassign, mprand, strtmp, mpfmt, mptoa, betom, mpitob, letom, mptole, mptoui, uitom, mptoi, utom, mpitou, vtom, mpitov, mpdigdiv, mpadd, mpsub, mpleft, mpright, mpitov, mpdiv, mpmul, mpmagcmp, mpmacmp, mpmagadd, mpmagsub, crtpre, crtinv, crtou, crtprefree, crtresfree – extended precision arithmetic

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <mp.h>

mpint* mpnew(int n)
void mpfree(mpint *b)
void mpsetminbits(int n)
void mpbits(mpint *b, int n)
void mpnorm(mpint *b)
mpint* mpcopy(mpint *b)
void mpassign(mpint *old, mpint *new)
mpint* mprand(int bits, void (*gen)(uchar*, int), mpint *b)
mpint* strtmp(char *buf, char **rptr, int base, mpint *b)
char* mptoa(mpint *b, int base, char *buf, int blen)
int mpfmt(Fmt*)
mpint* betom(char *buf, uint blen, mpint *b)
int mptobe(mpint *b, uchar *buf, uint blen, uchar **bufp)
mpint* letom(mpint *b, uint blen, mpint *b)
int mptole(mpint *b, uchar *buf, uint blen, uchar **bufp)
uint mptoui(mpint*)
mpint* uitom(uint, mpint*)
int mptoi(mpint*)

mpint* vtomp(vlong, mpint*)
vlong mptov(mpint*)

mpint* uvtomp(uvlong, mpint*)

uvlong mptouv(mpint*)
void mpadd(mpint *b1, mpint *b2, mpint *sum)
void mpmagadd(mpint *b1, mpint *b2, mpint *sum)
void mpsub(mpint *b1, mpint *b2, mpint *diff)
void mpmagsub(mpint *b1, mpint *b2, mpint *diff)
void mpleft(mpint *b, int shift, mpint *res)
void mpright(mpint *b, int shift, mpint *res)
void mpmul(mpint *b1, mpint *b2, mpint *prod)
void mpexp(mpint *b, mpint *e, mpint *m, mpint *res)
void mpmod(mpint *b, mpint *m, mpint *remainder)
void mpdiv(mpint *dividend, mpint *divisor, mpint *quotient, mpint *remainder)
int mpcmp(mpint *b1, mpint *b2)
int mpmagcmp(mpint *b1, mpint *b2)
void mpedgedgcd(mpint *a, mpint *b, mpint *d, mpint *x, mpint *y)
void mpinvert(mpint *b, mpint *m, mpint *res)
int mpsignif(mpint *b)

DESCRIPTION
These routines perform extended precision integer arithmetic. The basic type is mpint, which
points to an array of mpdigits, stored in little-endian order:

typedef struct mpint mpint;
struct mpint{
    int sign;  /* +1 or -1 */
    int size;  /* allocated digits */
    int top;  /* significant digits */
    mpdigit *p;
    char flags;
};
The sign of 0 is +1.
The size of mpdigit is architecture-dependent and defined in /$cputype/include/u.h.
Mpints are dynamically allocated and must be explicitly freed. Operations grow the array of dig-
its as needed.
In general, the result parameters are last in the argument list.
Routines that return an mpint will allocate the mpint if the result parameter is nil. This
includes strtomp, itomp, uitomp, and btomp. These functions, in addition to mpnew and mpcopy,
will return nil if the allocation fails.
Input and result parameters may point to the same mpint. The routines check and copy where
necessary.
\textit{Mpnew} creates an \texttt{mpint} with an initial allocation of \textit{n} bits. If \textit{n} is zero, the allocation will be whatever was specified in the last call to \textit{mpsetminbits} or to the initial value, 1056. \textit{Mpfree} frees an \texttt{mpint}. \textit{Mpbits} grows the allocation of \texttt{b} to fit at least \textit{n} bits. If \texttt{b->top} doesn’t cover \textit{n} bits, \textit{mpbits} increases it to do so. Unless you are writing new basic operations, you can restrict yourself to \texttt{mpnew(0)} and \texttt{mpfree(b)}.

\textit{Mpnorm} normalizes the representation by trimming any high order zero digits. All routines except \textit{mpbits} return normalized results.

\textit{Mprand} creates a new \texttt{mpint} with the same value as \texttt{b} while \textit{mpassign} sets the value of \textit{new} to be that of \textit{old}.

\textit{Mprand} creates an \textit{n} bit random number using the generator \textit{gen}. \textit{Gen} takes a pointer to a string of uchar's and the number to fill in.

\textit{Strtomp} and \textit{mptoa} convert between ASCII and \texttt{mpint} representations using the base indicated. Only the bases 10, 16, 32, and 64 are supported. Anything else defaults to 16. \textit{Strtomp} skips any leading spaces or tabs. \textit{Strtomp}'s scan stops when encountering a digit not valid in the base. If \texttt{rptr} is not zero, \texttt{*rptr} is set to point to the character immediately after the string converted. If the parse terminates before any digits are found, \textit{strtomp} returns \texttt{nil}. \textit{Mptoascii} returns a pointer to the filled buffer. If the parameter \texttt{buf} is \texttt{nil}, the buffer is allocated. \textit{Mpfmt} can be used with \textit{fmtinstall(2)} and \textit{print(2)} to print hexadecimal representations of \texttt{mpints}. The conventional verb is \texttt{B}, for which \texttt{mp.h} provides a pragma.

\textit{Mptobe} and \textit{mptole} convert an \texttt{mpint} to a byte array. The former creates a big endian representation, the latter a little endian one. If the destination \texttt{buf} is \texttt{nil}, it specifies the buffer of length \texttt{blen} for the result. If the representation is less than \texttt{blen} bytes, the rest of the buffer is zero filled. If \texttt{buf} is \texttt{nil}, then a buffer is allocated and a pointer to it is deposited in the location pointed to by \texttt{bufp}. Sign is ignored in these conversions, i.e., the byte array version is always positive.

\textit{Betomp} and \textit{letomp} convert from a big or little endian byte array at \texttt{buf} of length \texttt{blen} to an \texttt{mpint}. If \texttt{b} is not \texttt{nil}, it refers to a preallocated \texttt{mpint} for the result. If \texttt{b} is \texttt{nil}, a new integer is allocated and returned as the result.

The integer conversions are:

- \texttt{mptoui} \texttt{mpint->unsigned int}
- \texttt{uitomp} \texttt{unsigned int->mpint}
- \texttt{mptoi} \texttt{mpint->int}
- \texttt{itomp} \texttt{int->mpint}
- \texttt{mptouil} \texttt{mpint->unsigned vlong}
- \texttt{uvtomp} \texttt{unsigned vlong->mpint}
- \texttt{mptov} \texttt{mpint->vlong}
- \texttt{vtomp} \texttt{vlong->mpint}

When converting to the base integer types, if the integer is too large, the largest integer of the appropriate sign and size is returned.

The mathematical functions are:

- \texttt{mpadd} \texttt{sum = b1 + b2.}
- \texttt{mpmagadd} \texttt{sum = abs(b1) + abs(b2).}
- \texttt{mpsub} \texttt{diff = b1 - b2.}
- \texttt{mpmagsub} \texttt{diff = abs(b1) - abs(b2).}
- \texttt{mpleft} \texttt{res = b<<shift.}
- \texttt{mpright} \texttt{res = b>>shift.}
- \texttt{mpmul} \texttt{prod = b1*b2.}
- \texttt{mpevp} \texttt{if m is nil, res = b**e. Otherwise, res = b**e mod m.}
- \texttt{mpmod} \texttt{remainder = b % m.}
- \texttt{mpdiv} \texttt{quotient = dividend/divisor. remainder = dividend % divisor.}
- \texttt{mpcmp} \texttt{returns -1, 0, or +1 as b1 is less than, equal to, or greater than b2.}
- \texttt{mpmagcmp} \texttt{the same as mpcmp but ignores the sign and just compares magnitudes.}

\textit{Mextendedgcd} computes the greatest common denominator, \textit{d}, of \textit{a} and \textit{b}. It also computes \textit{x} and \textit{y} such that \texttt{a*x + b*y = d}. Both \textit{a} and \textit{b} are required to be positive. If called with negative arguments, it will return a \texttt{gcd} of 0.
\textit{Mpinverse} computes the multiplicative inverse of $b$ mod $m$.

\textit{Mpsignif} returns the number of significant bits in $b$. \textit{Mplowbits0} returns the number of consecutive zero bits at the low end of the significant bits. For example, for 0x14, \textit{mpsignif} returns 5 and \textit{mplowbits0} returns 2. For 0, \textit{mpsignif} and \textit{mplowbits0} both return 0.

The remaining routines all work on arrays of \textit{mpdigit} rather than \textit{mpint}'s. They are the basis of all the other routines. They are separated out to allow them to be rewritten in assembler for each architecture. There is also a portable C version for each one.

\begin{verbatim}
mpdigdiv quotient = dividend[0:1] / divisor.
mpvecadd sum[0:alen] = a[0:alen-1] + b[0:blen-1]. We assume
               alen >= blen and that sum has room for alen+1 digits.
mpvecsub diff[0:alen-1] = a[0:alen-1] - b[0:blen-1]. We assume
               that alen >= blen and that diff has room for alen digits.
mpvecdigmuladd p[0:n] += m * b[0:n-1]. This multiplies an array of digits times
               a scalar and adds it to another array. We assume p has room for n+1 dig-
               its.
mpvecdigmulsub p[0:n] -= m * b[0:n-1]. This multiplies an array of digits times
               a scalar and subtracts it from another array. We assume p has room for n+1 dig-
               its. It returns +1 if the result is positive and -1 if negative.
mpvecmul p[0:alen*blen] = a[0:alen-1] * b[0:blen-1]. We assume
               that p has room for alen*blen+1 digits.
mpveccmp This returns -1, 0, or +1 as a - b is negative, 0, or positive.
\end{verbatim}

\textit{mptwo}, \textit{mpone} and \textit{mpzero} are the constants 2, 1 and 0. These cannot be freed.

\textbf{Chinese remainder theorem}

When computing in a non-prime modulus, $n$, it is possible to perform the computations on the residues modulo the prime factors of $n$ instead. Since these numbers are smaller, multiplication and exponentiation can be much faster.

\textit{Crtin} computes the residues of $x$ and returns them in a newly allocated structure:

\begin{verbatim}
typedef struct CRTres CRTres;
{
  int n;  /* number of residues */
  mpint *r[n];  /* residues */
};
\end{verbatim}

\textit{Crtout} takes a residue representation of a number and converts it back into the number. It also frees the residue structure.

\textit{Crepre} saves a copy of the factors and precomputes the constants necessary for converting the residue form back into a number modulo the product of the factors. It returns a newly allocated structure containing values.

\textit{Crtprefree} and \textit{crtresfree} free \textit{CRTpre} and \textit{CRTres} structures respectively.

\textbf{SOURCE}

/sys/src/libmp
NAME
muldiv, umuldiv - high-precision multiplication and division

SYNOPSIS
#include <u.h>
#include <libc.h>
long muldiv(long a, long b, long c)
ulong umuldiv(ulong a, ulong b, ulong c)

DESCRIPTION
Muldiv returns a*b/c, using a vlong to hold the intermediate result. Umuldiv is the equivalent for unsigned integers. They can be used to scale integer values without worry about overflowing the intermediate result.

On some architectures, these routines can generate a trap if the final result does not fit in a long or ulong; on others they will silently truncate.
NAME
   NaN, Inf, isNaN, isInf – not-a-number and infinity functions

SYNOPSIS
   #include <u.h>
   #include <libc.h>
   double NaN(void)
   double Inf(int)
   int   isNaN(double)
   int   isInf(double, int)

DESCRIPTION
   The IEEE floating point standard defines values called 'not-a-number' and positive and negative
   'infinity'. These values can be produced by such things as overflow and division by zero. Also, the
   library functions sometimes return them when the arguments are not in the domain, or the result
   is out of range. By default, manipulating these values may cause a floating point exception on
   some processors but setfcr (see getfcr(2)) can change that behavior.

   NaN returns a double that is not-a-number. isNaN returns true if its argument is not-a-number.
   Inf(i) returns positive infinity if i is greater than or equal to zero, else negative infinity. IsInf
   returns true if its first argument is infinity with the same sign as the second argument.

SOURCE
   /sys/src/libc/port/nan.c

SEE ALSO
   getfcr(2)
NAME

ndbopen, ndbcat, ndbchanged, ndbclose, ndbreopen, ndbsearch, ndbsnext, ndbgetvalue, ndbfree, 
ipatr, ndbgetipaddr, ndbipinfo, csipinfo, ndbhash, ndbparse, csgetvalue, ndbfindattr, ndsquery, 
ndbdiscard, ndbconcatenate, ndbreorder, ndbsubstitute, ndbgetval, csgetval, ndblookval – network database

SYNOPSIS

#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ndb.h>

Ndb* ndbopen(char *file)
Ndb* ndbcat(Ndb *db1, Ndb *db2)
int ndbchanged(Ndb *db)
int ndbreopen(Ndb *db)
void ndbclose(Ndb *db)
Ndbtuple* ndbsearch(Ndb *db, Ndbtuple *s, char *attr, char *val)
Ndbtuple* ndbsnext(Ndbtuple *s, char *attr, char *val)
char* ndbgetvalue(Ndb *db, Ndbtuple *s, char *attr, char *val, 
char *rattr, Ndbtuple **tp)
char* csgetvalue(char *netroot, char *attr, char *val, 
char *rattr, Ndbtuple **tp)
char* ipatr(char *name)
Ndbtuple* ndbgetipaddr(Ndb *db, char *sys);
Ndbtuple* ndbipinfo(Ndb *db, char *attr, char *val, char **attrs, 
int nattr)
Ndbtuple* csipinfo(char *netroot, char *attr, char *val, 
char **attrs, int nattr)
ulong ndbhash(char *val, int hlen)
Ndbtuple* ndbparse(Ndb *db)
Ndbtuple* dnsquery(char *netroot, char *domainname, char *type)
Ndbtuple* ndbfindattr(Ndbtuple *entry, Ndbtuple *line, char *attr)
void ndbfree(Ndbtuple *db)
Ndbtuple* ndbdiscard(Ndbtuple *t, Ndbtuple *a)
Ndbtuple* ndbconcatenate(Ndbtuple *a, Ndbtuple *b)
Ndbtuple* ndbreorder(Ndbtuple *t, Ndbtuple *a)
Ndbtuple* ndbsubstitute(Ndbtuple *t, Ndbtuple *from, Ndbtuple *to)
void ndbsetmalloctag(Ndbtuple *t, uintptr tag)

DESCRIPTION

These routines are used by network administrative programs to search the network database. 
They operate on the database files described in ndb(6).

Ndbsopen opens the database file and calls malloc(2) to allocate a buffer for it. If file is zero, all 
network database files are opened.

Ndbscat concatenates two open databases. Either argument may be nil.

Ndbsreopen throws out any cached information for the database files associated with db and 
reopens the files.
Ndbclose closes any database files associated with db and frees all storage associated with them.

Ndbsearch and ndbsnext search a database for an entry containing the attribute/value pair, attr= val. Ndbsearch is used to find the first match and ndbsnext is used to find each successive match. On a successful search both return a linked list of Ndbtuple structures acquired by malloc(2) that represent the attribute/value pairs in the entry. On failure they return zero.

```c
typedef struct Ndbtuple Ndbtuple;
struct Ndbtuple {
    char  attr[Ndbalen];
    char  *val;
    Ndbtuple *entry;
    Ndbtuple *line;
    ulong  ptr;  /* for the application; starts 0 */
    char  valbuf[Ndbvlen];  /* initial allocation for val */
};
```

The entry pointers chain together all pairs in the entry in a null-terminated list. The line pointers chain together all pairs on the same line in a circular list. Thus, a program can implement 2 levels of binding for pairs in an entry. In general, pairs on the same line are bound tighter than pairs on different lines.

The argument s of ndbsearch has type Nds and should be pointed to valid storage before calling ndbsearch, which will fill it with information used by ndbsnext to link successive searches. The structure Nds looks like:

```c
typedef struct Ndb Ndb;
struct Ndb {
    Ndb  *db;  /* data base file being searched */
    ...
    Ndbtuple *t;  /* last attribute value pair found */
};
```

The t field points to the pair within the entry matched by the ndbsearch or ndbsnext.

Ndbgetvalue searches the database for an entry containing not only an attribute/value pair, attr= val, but also a pair with the attribute rattr. If successful, it returns a malloced copy of the NUL-terminated value associated with rattr. If tp is non nil, *tp will point to the entry. Otherwise the entry will be freed.

C getvalue is like ndbgetvalue but queries the connection server instead of looking directly at the database. Its first argument specifies the network root to use. If the argument is 0, it defaults to "/net".

Ndbfree frees a list of tuples returned by one of the other routines.

Ipattr takes the name of an IP system and returns the attribute it corresponds to:

```c
dom  domain name
ip   Internet number
sys  system name
```

Ndbgetipaddr looks in db for an entry matching sys as the value of a sys= or dom= attribute/value pair and returns all IP addresses in the entry. If sys is already an IP address, a tuple containing just that address is returned.

Ndbipinfo looks up Internet protocol information about a system. This is an IP aware search. It looks first for information in the system’s database entry and then in the database entries for any IP subnets or networks containing the system. The system is identified by the attribute/value pair, attr= val. Ndbipinfo returns a list of tuples whose attributes match the attributes in the n element array attrs. If any attrs begin with @, the @ is excluded from the attribute name, but causes any corresponding value returned to be a resolved IP address(es), not a name. For example, consider the following database entries describing a network, a subnet, and a system.

```c
ipnet=big  ip=10.0.0.0
dns=dns.big.com
smtp=smtp.big.com
```
ipnet=dept ip=10.1.1.0 ipmask=255.255.255.0
smtp=smtpl.big.com
ip=10.1.1.4 dom=x.big.com
bootf=/386/9pc
Calling

ndbipinfo(db, "dom", "x.big.com", ["bootf" "smtp" "dns"], 3)
will return the tuples bootf=/386/9pc, smtp=smtp1.big.com, and dns=dns.big.com.
Cisco is to ndbipinfo as csetval is to ndbgetval.
The next three routines are used by programs that create the hash tables and database files. Ndbhash computes a hash offset into a table of length hlen for the string val. Ndbparse reads and parses the next entry from the database file. Multiple calls to ndbparse parse sequential entries in the database file. A zero is returned at end of file.
Dnsquery submits a query about domainname to the ndb/dns mounted at netroot/dns. It returns a linked list of Ndbtuple’s representing a single database entry. The tuples are logically arranged into lines using the line field in the structure. The possible type’s of query are and the attributes on each returned tuple line is:
ip find the IP addresses. Returns domain name (dom) and ip address (ip)
mx look up the mail exchangers. Returns preference (pref) and exchanger (mx)
ptr do a reverse query. Here domainname must be an ASCII IP address. Returns reverse name (ptr) and domain name (dom)
cname get the system that this name is a nickname for. Returns the nickname (dom) and the real name (cname)
soa return the start of area record for this field. Returns area name (dom), primary name server (ns), serial number (serial), refresh time in seconds (refresh), retry time in seconds (retry), expiration time in seconds (expire), and minimum time to lie (ttl).
ns name servers. Returns domain name (dom) and name server (ns)
Ndbfindattr searches entry for the tuple with attribute attr and returns a pointer to the tuple. If line points to a particular line in the entry, the search starts there and then wraps around to the beginning of the entry.
All of the routines provided to search the database provide an always consistent view of the relevant files. However, it may be advantageous for an application to read in the whole database using ndbopen and ndbparse and provide its own search routines. The ndbchanged routine can be used by the application to periodically check for changes. It returns zero if none of the files comprising the database have changes and non-zero if they have.
Finally, a number of routines are provided for manipulating tuples.
Ndbdiscard removes attr/val pair a from tuple t and frees it. If a isn’t in t it is just freed.
Ndbconcatenate concatenates two tuples and returns the result. Either or both tuples may be nil.
Ndbreorder reorders a tuple t to make the line containing attr/val pair a first in the entry and making a first in its line.
Ndbsubstitute replaces a single att/val pair from in t with the tuple to. All att/val pairs in to end up on the same line. from is freed.
Ndbsetmalloctag sets the malloc tag (see setmalloctag in malloc(2)) of each tuple in the list t to tag.
FILES
/lib/ndb directory of network database files
SOURCE
/sys/src/libndb
SEE ALSO
ndb(6), ndb(8)
**DIAGNOSTICS**

`Ndbgetvalue`, `csgetvalue`, and `ndblookvalue` set `errstr` to `buffer too short` if the buffer provided isn't long enough for the returned value.

**BUGS**

`Ndbgetval`, `csgetval`, and `ndblookval` are deprecated versions of `ndbgetvalue`, `csgetvalue`, and `ndblookvalue`. They expect a fixed 64 byte long result buffer and existed when the values of a `Ndbtuple` structure were fixed length.
**NAME**
notify, noted, atnotify - handle asynchronous process notification

**SYNOPSIS**
```c
#include <u.h>
#include <libc.h>
int notify(void (*f)(void*, char*))
int noted(int v)
int atnotify(int (*f)(void*, char*), int in)
```

**DESCRIPTION**
When a process raises an exceptional condition such as dividing by zero or writing on a closed pipe, a note is posted to communicate the exception. A note may also be posted by a write (see read(2)) to the process's /proc/n(note file or to the /proc/m/notes file of a process in the same process group (see proc(3)). When the note is received the behavior of the process depends on the origin of the note. If the note was posted by an external process, the process receiving the note exits; if generated by the system the note string, preceded by the name and id of the process and the string "suicide: ", is printed on the process's standard error file and the process is suspended in the Broken state for debugging.

These default actions may be overridden. The notify function registers a notification handler to be called within the process when a note is received. The argument to notify replaces the previous handler, if any. An argument of zero cancels a previous handler, restoring the default action. A fork(2) system call leaves the handler registered in both the parent and the child; exec(2) restores the default behavior. Handlers may not perform floating point operations.

After a note is posted, the handler is called with two arguments: the first is a pointer to a Ureg structure (defined in /subsys/include/ureg.h) giving the current values of registers; the second is a pointer to the note itself, a null-terminated string with no more than ERRLEN characters in it including the terminal NUL. The Ureg argument is usually not needed; it is provided to help recover from traps such as floating point exceptions. Its use and layout are machine- and system-specific.

A notification handler must finish either by exiting the program or by calling noted; if the handler returns the behavior is undefined and probably erroneous. Until the program calls noted, any further externally-generated notes (e.g., hangup or alrm) will be held off, and any further notes generated by erroneous behavior by the program (such as divide by zero) will kill the program. The argument to noted defines the action to take: NDFLT instructs the system to perform the default action as if the handler had never been registered; NCONT instructs the system to resume the process at the point it was notified. In neither case does noted return to the handler. If the note interrupted an incomplete system call, that call returns an error (with error string interrupted) after the process resumes. A notification handler can also jump out to an environment set up with setjmp using the notejmp function (see setjmp(2)), which is implemented by modifying the saved state and calling noted(NCONT).

Regardless of the origin of the note or the presence of a handler, if the process is being debugged (see proc(3)) the arrival of a note puts the process in the Stopped state and awakens the debugger.

**Atnotify**
Rather than using the system calls notify and noted, most programs should use atnotify to register notification handlers. The parameter in is non-zero to register the function f, and zero to cancel registration. A handler must return a non-zero number if the note was recognized (and resolved); otherwise it must return zero. When the system posts a note to the process, each handler registered with atnotify is called with arguments as described above until one of the handlers returns non-zero. Then noted is called with argument NCONT. If no registered function returns non-zero, atnotify calls noted with argument NDFLT.

**APE**
Noted has two other possible values for its argument. NSAVE returns from the handler and clears the note, enabling the receipt of another, but does not return to the program. Instead it starts a new handler with the same stack, stack pointer, and arguments as the original, at the address
recorded in the program counter of the Ureg structure. Typically, the program counter will be
overridden by the first note handler to be the address of a separate function; NSAVE is then a
‘trampoline’ to that handler. That handler may executed noted(NRSTR) to return to the origi-
nal program, usually after restoring the original program counter. NRSTR is identical to NCONT
except that it can only be executed after an NSAVE. NSAVE and NRSTR are designed to improve
the emulation of signals by the ANSI C/POSIX environment; their use elsewhere is discouraged.

Notes
The set of notes a process may receive is system–dependent, but there is a common set that
includes:

<table>
<thead>
<tr>
<th>Note</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt</td>
<td>user interrupt (DEL key)</td>
</tr>
<tr>
<td>hangup</td>
<td>I/O connection closed</td>
</tr>
<tr>
<td>alarm</td>
<td>alarm expired</td>
</tr>
<tr>
<td>sys: breakp oint</td>
<td>breakpoint instruction</td>
</tr>
<tr>
<td>sys: bad address</td>
<td>system call address argument out of range</td>
</tr>
<tr>
<td>sys: odd address</td>
<td>system call address argument unaligned</td>
</tr>
<tr>
<td>sys: bad sys call</td>
<td>system call number out of range</td>
</tr>
<tr>
<td>sys: odd stack</td>
<td>system call user stack unaligned</td>
</tr>
<tr>
<td>sys: write on closed pipe</td>
<td>write on closed pipe</td>
</tr>
<tr>
<td>sys: fp: fptrap</td>
<td>floating point exception</td>
</tr>
<tr>
<td>sys: trap: trap</td>
<td>other exception (see below)</td>
</tr>
</tbody>
</table>

The notes prefixed sys: are generated by the operating system. They are suffixed by the user
program counter in format pc=0x1234. If the note is due to a floating point exception, just
before the pc is the address of the offending instruction in format fppc=0x1234. Notes are
limited to ERRLEN bytes; if they would be longer they are truncated but the pc is always reported
correctly.

The types and syntax of the trap and fptrap portions of the notes are machine–dependent.

SOURCE
/sys/src/libc/9syscall
/sys/src/libc/port/atnotify.c

SEE ALSO
intro(2), notejmp in setjmp(2)

BUGS
Since exec(2) discards the notification handler, there is a window of vulnerability to notes in a new
process.
NAME

objtype, readobj, objtraverse, isar, nextar, readar - object file interpretation functions

SYNOPSIS

#include <u.h>
#include <libc.h>
#include <bio.h>
#include <mach.h>

int objtype(Biobuf *bp, char **name)
int readobj(Biobuf *bp, int objtype)
void objtraverse(void(*)(Sym*, void*), void*)
int isar(Biobuf *bp)
int nextar(Biobuf *bp, int offset, char *buf)
int readar(Biobuf *bp, int objtype, int end)

DESCRIPTION

These functions provide machine-independent access to object files in a directory or an archive. Mach(2) and symbol(2) describe additional library functions for interpreting executable files and executing images.

Object files contain no formal symbol table; instead, references to symbols must be extracted from the encoded object representation and resolved. The resulting symbol information is loaded into a dummy symbol table where it is available for processing by an application. The organization of the dummy symbol table is identical to that produced by the loader and described in symbol(2) and a.out(6): a vector of Sym data structures defining the name, type and relative offset of each symbol.

Objtype reads the header at the current position of the file associated with bp (see Bio(2)) to see if it is an intermediate object file. If it is, a code indicating the architecture type of the file is returned and the second argument, if it is non-zero, is set pointing to a string describing the type of the file. If the header does not indicate an object file, -1 is returned. The header may be at the start of an object file or at the beginning of an archive member. The file is rewound to its starting position after decoding the header.

Readdobj constructs a symbol table for the object file associated with bp. The second argument contains the type code produced by function objtype. The file must be positioned at the start of the object file. Each invocation of readdobj destroys the symbol definitions for any previous file.

Objtraverse scans the symbol table previously built by readdobj or readar. Objtraverse requires two arguments: the address of a call-back function and a generic pointer. The call-back function is invoked once for each symbol in the symbol table with the address of a Sym data structure as the first argument and the generic pointer as the second.

Isar reads the header at the current point in the file associated with bp and returns 1 if it is an archive or zero otherwise. The file is positioned at the end of the archive header and at the beginning of the first member of the archive.

Nextar extracts information describing the archive member stored at offset in the file associated with bp. If the header describing the member can be extracted and decoded, the size of the member is returned. Adding this value to offset yields the offset of the beginning of the next member in the archive. On return the input file is positioned at the end of the member header and the name of the member is stored in buf, a buffer of SARMAME characters. If there are no more members, nextar returns zero; a negative return indicates a missing or malformed header.

Readar constructs the symbol table of the object file stored at the current position in the archive associated with bp. This function operates exactly as readdobj; the only difference is the extra argument, end, specifying the offset to the beginning of the next member in the archive. Readar leaves the file positioned at that point.

SOURCE

/sys/src/libmach
SEE ALSO
   mach(2), symbol(2), bio(2), a.out(6)

DIAGNOSTICS
   These routines set errstr.
NAME
open, create, close - open a file for reading or writing, create file

SYNOPSIS
#include <u.h>
#include <libc.h>
int open(char *file, int omode)
int create(char *file, int omode, ulong perm)
int close(int fd)

DESCRIPTION
Open opens the file for I/O and returns an associated file descriptor. Omode is one of OREAD, OWRITE, ORDWR, or OEXEC, asking for permission to read, write, read and write, or execute, respectively. In addition, there are three values that can be ORed with the omode: OTRUNC says to truncate the file to zero length before opening it; OEXEC says to close the file when an exec(2) or excecl system call is made; and ORCLOSE says to remove the file when it is closed (by everyone who has a copy of the file descriptor). Open fails if the file does not exist or the user does not have permission to open it for the requested purpose (see stat(2) for a description of permissions). The user must have write permission on the file if the OTRUNC bit is set. For the open system call (unlike the implicit open in exec(2)), OEXEC is actually identical to OREAD.

Create creates a new file or prepares to rewrite an existing file, opens it according to omode (as described for open), and returns an associated file descriptor. If the file is new, the owner is set to the userid of the creating process group; the group to that of the containing directory; the permissions to perm ANDed with the permissions of the containing directory. If the file already exists, it is truncated to 0 length, and the permissions, owner, and group remain unchanged. The created file is a directory if the DMDIR bit is set in perm, an exclusive-use file if the DMEXCL bit is set, and an append-only file if the DMAPPEND bit is set. Exclusive-use files may be open for I/O by only one client at a time, but the file descriptor may become invalid if no I/O is done for an extended period; see open(5).

Create fails if the path up to the last element of file cannot be evaluated, if the user doesn't have write permission in the final directory, if the file already exists and does not permit the access defined by omode, or if there are no free file descriptors. In the last case, the file may be created even when an error is returned. If the file is new and the directory in which it is created is a union directory (see intro(2)) then the constituent directory where the file is created depends on the structure of the union: see bind(2).

Since create may succeed even if the file exists, a special mechanism is necessary for those applications that require an atomic create operation. If the OEXCL (0x1000) bit is set in the mode for a create, the call succeeds only if the file does not already exist; see open(5) for details.

Close closes the file associated with a file descriptor. Provided the file descriptor is a valid open descriptor, close is guaranteed to close it; there will be no error. Files are closed automatically upon termination of a process; close allows the file descriptor to be reused.

SOURCE
/sys/src/libc/9syscall

SEE ALSO
intro(2), bind(2), stat(2)

DIAGNOSTICS
These functions set errstr.
NAME
peror, syslog, sysfatal - system error messages

SYNOPSIS
#include <u.h>
#include <libc.h>
void perror(char *s)
void syslog(int cons, char *logname, char *fmt, ...)
void sysfatal(char *fmt, ...)

DESCRIPTION
Perror produces a short error message on the standard error file describing the last error encountered during a call to the system. First the argument string s is printed, then a colon, then the message and a newline. If s is nil, only the error message and newline are printed.

Syslog logs messages in the file named by logname in the directory /sys/log; the file must already exist and should be append-only. Logname must contain no slashes. The message is a line with several fields: the name of the machine writing the message; the date and time; the message specified by the print(2) format fmt and any following arguments; and a final newline. If cons is set or the log file cannot be opened, the message is also printed on the system console. Syslog can be used safely in multi-threaded programs.

Sysfatal prints to standard error the name of the running program, a colon and a space, the message described by the print(2) format string fmt and subsequent arguments, and a newline. It then calls exits(2) with the formatted message as argument. The program’s name is the value of argv0, which will be set if the program uses the arg(2) interface to process its arguments. If argv0 is null, it is ignored and the following colon and space are suppressed.

SOURCE
/sys/src/libc/port/perror.c
/sys/src/libc/9sys/syslog.c
/sys/src/libc/9sys/sysfatal.c

SEE ALSO
intro(2), errstr(2), the %r format in print(2)

BUGS
Perror is a holdover; the %r format in print(2) is preferred.
NAME
  pipe - create an interprocess channel

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  int pipe(int fd[2])

DESCRIPTION
  Pipe creates a buffered channel for interprocess I/O communication. Two file descriptors are
  returned in fd. Data written to fd[1] is available for reading from fd[0] and data written to
  fd[0] is available for reading from fd[1].

  After the pipe has been established, cooperating processes created by subsequent fork(2) calls
  may pass data through the pipe with read and write calls. The bytes placed on a pipe by one write
  are contiguous even if many processes are writing. Write boundaries are preserved: each read ter-
  minates when the read buffer is full or after reading the last byte of a write, whichever comes first.

  The number of bytes available to a read(2) is reported in the Length field returned by fstat or
  dirfstat on a pipe (see stat(2)).

  When all the data has been read from a pipe and the writer has closed the pipe or exited, read(2)
  will return 0 bytes. Writes to a pipe with no reader will generate a note sys: write on closed pipe.

SOURCE
  /sys/src/libc/9syscall

SEE ALSO
  intro(2), read(2), pipe(3)

DIAGNOSTICS
  Sets errstr.

BUGS
  If a read or a write of a pipe is interrupted, some unknown number of bytes may have been trans-
  ferred.

  When a read from a pipe returns 0 bytes, it usually means end of file but is indistinguishable from
  reading the result of an explicit write of zero bytes.
NAME
eplumb, plumbfree, plumbopen, plumbsend, plumbsendtext, plumblookup, plumbpack, plumbpackattr, plumbaddrattr, plumbdelattr, plumbrecv, plumbunpack, plumbunpackpartial, plumbpackattr, Plumbmsg – plumb messages

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <plumb.h>

int plumbopen(char *port, int omode)
int plumbsend(int fd, Plumbmsg *m)
int plumbsendtext(int fd, char *src, char *dst, char *wdir, char *data)
void plumbfree(Plumbmsg *m)
Plumbmsg* plumbrecv(int fd)
typedef struct Plumbmsg
{ char *src;
  char *dst;
  char *wdir;
  char *type;
  Plumbattr *attr;
  int ndata;
  char *data;
} Plumbmsg;

typedef struct Plumbattr
{ char *name;
  char *value;
  Plumbattr *next;
} Plumbattr;

Plumbopen opens the named plumb port, using open(2) mode omode. If port begins with a slash, it is taken as a literal file name; otherwise plumbopen searches for the location of the plumber(4) service and opens the port there.

DESCRIPTION
These routines manipulate plumb(6) messages, transmitting them, receiving them, and converting them between text and these data structures:

typedef
struct Plumbmsg
{ char *src;
  char *dst;
  char *wdir;
  char *type;
  Plumbattr *attr;
  int ndata;
  char *data;
} Plumbmsg;

typedef
struct Plumbattr
{ char *name;
  char *value;
  Plumbattr *next;
} Plumbattr;

For programs using the event(2) interface, eplumb registers, using the given key, receipt of messages from the named port.
Plumbsend formats and writes message \(m\) to the file descriptor \(fd\), which will usually be the result of plumbopen(“send”, OWRITE). Plumbsendtext is a simplified version for text-only messages; it assumes type is text, sets attr to nil, and sets ndata to strlen(data).

Plumbfree frees all the data associated with the message \(m\), all the components of which must therefore have been allocated with malloc(2).

Plumbrecv returns the next message available on the file descriptor \(fd\), or nil for error.

Plumbpack encodes message \(m\) as a character string in the format of plumb(6), setting *np to the length in bytes of the string. Plumbunpack does the inverse, translating the \(n\) bytes of buf into a Plumbmsg.

Plumbunpackpartial enables unpacking of messages that arrive in pieces. The first call to plumbunpackpartial for a given message must be sufficient to unpack the header; subsequent calls permit unpacking messages with long data sections. For each call, buf points to the beginning of the complete message received so far, and \(n\) reports the total number of bytes received for that message. If the message is complete, the return value will be as in plumbunpack. If not, and morep is not null, the return value will be nil and *morep will be set to the number of bytes remaining to be read for this message to be complete (recall that the byte count is in the header). Those bytes should be read by the caller, placed at location buf+\(n\), and the message unpacked again. If an error is encountered, the return value will be nil and *morep will be zero.

Plumbpackattr converts the list \(a\) of Plumbattr structures into a null-terminated string. If an attribute value contains white space, quote characters, or equal signs, the value will be quoted appropriately. A newline character will terminate processing. Plumbunpackattr converts the null-terminated string \(a\) back into a list of Plumbattr structures.

Plumblookup searches the Plumbattr list \(a\) for an attribute with the given name and returns the associated value. The returned string is the original value, not a copy. If the attribute has no value, the returned value will be the empty string; if the attribute does not occur in the list at all, the value will be nil.

Plumbaddattr appends the new Plumbattr (which may be a list) to the attribute list \(a\) and returns the new list. Plumbattr searches the list \(a\) for the first attribute with name name and deletes it from the list, returning the resulting list. Plumbdelattr is a no-op if no such attribute exists.

SOURCE
/sys/src/libplumb

SEE ALSO
plumb(1), event(2), plumber(4), plumb(6)

DIAGNOSTICS
When appropriate, including when a plumbsend fails, these routine set errstr.
NAME
poolalloc, poolallocalign, poolfree, poolmsize, poolrealloc, poolcompact, poolcheck, poolblockcheck, poldump - general memory management routines

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <pool.h>

void* poolalloc(Pool* pool, ulong size)
void* poolallocalign(Pool* pool, ulong size, ulong align, long offset, ulong span)

void poolfree(Pool* pool, void* ptr)
ulong poolmsize(Pool* pool, void* ptr)

void* poolrealloc(Pool* pool, void* ptr, ulong size)
void poolcompact(Pool* pool)
void poolcheck(Pool* pool)
void poolblockcheck(Pool* pool, void* ptr)
void poldump(Pool* pool);

DESCRIPTION
These routines provide a general memory management facility. Memory is retrieved from a coarser allocator (e.g. sbrk or the kernel’s xmalloc) and then allocated to callers. The routines are locked and thus may safely be used in multiprocess programs.

Poolalloc attempts to allocate a block of size size; it returns a pointer to the block when successful and nil otherwise. The call poolalloc(0) returns a non-nil pointer. Poolfree returns an allocated block to the pool. It is an error to free a block more than once or to free a pointer not returned by poolalloc. The call poolfree(nil) is legal and is a no-op.

Poolallocalign attempts to allocate a block of size size with the given alignment constraints. If align is non-zero, the returned pointer is aligned to be equal to offset modulo align. If span is non-zero, the n byte block allocated will not span a span-byte boundary.

Poolrealloc attempts to resize to nsize bytes the block associated with ptr, which must have been previously returned by poolalloc or poolrealloc. If the block’s size can be adjusted, a (possibly different) pointer to the new block is returned. The contents up to the lesser of the old and new sizes are unchanged. After a successful call to poolrealloc, the return value should be used rather than ptr to access the block. If the request cannot be satisfied, poolrealloc returns nil, and the old pointer remains valid.

When blocks are allocated, there is often some extra space left at the end that would usually go unused. Poolmsize grows the block to encompass this extra space and returns the new size.

The poolblockcheck and poolcheck routines validate a single allocated block or the entire pool, respectively. They call panic (see below) if corruption is detected. Poldump prints a summary line for every block in the pool, using the print function (see below).

The Pool structure itself provides much of the setup interface.

typedef struct Pool Pool;
struct Pool {
    char* name;
    ulong maxsize; /* of entire Pool */
    ulong cursize; /* of Pool */
    ulong curfree; /* total free bytes in Pool */
    ulong curalloc; /* total allocated bytes in Pool */
    ulong minarena; /* smallest size of new arena */
    ulong quantum; /* allocated blocks should be multiple of */
    ulong minblock; /* smallest newly allocated block */
    int flags;
};
int nfree;      /* number of calls to free */
int lastcompact; /* nfree at time of last poolcompact */
void* (*alloc)(ulong);
int (*merge)(void*, void*);
void (*move)(void* from, void* to);
void (*lock)(Pool*);
void (*unlock)(Pool*);
void (*print)(Pool*, char*, ...);
void (*panic)(Pool*, char*, ...);
void (*logstack)(Pool*);
void* private;

enum { /* flags */
    POOL_ANTAGONISM = 1<<0,
    POOL_PARANOIA = 1<<1,
    POOL_VERBOSITY = 1<<2,
    POOL_DEBUGGING = 1<<3,
    POOL_LOGGING = 1<<4,
    POOL_TOLERANCE = 1<<5,
    POOL_NOREUSE = 1<<6,
};

The pool obtains arenas of memory to manage by calling the the given alloc routine. The total number of requested bytes will not exceed maxsize. Each allocation request will be for at least minarena bytes.

When a new arena is allocated, the pool routines try to merge it with the surrounding arenas, in an attempt to combat fragmentation. If merge is non-nil, it is called with the addresses of two blocks from alloc that the pool routines suspect might be adjacent. If they are not mergeable, merge must return zero. If they are mergeable, merge should merge them into one block in its own bookkeeping and return non-zero.

To ease fragmentation and make block reuse easier, the sizes requested of the pool routines are rounded up to a multiple of quantum before the carrying out requests. If, after rounding, the block size is still less than minblock bytes, minblock will be used as the block size.

Poolcompact defragments the pool, moving blocks in order to aggregate the free space. Each time it moves a block, it notifies the move routine that the contents have moved. At the time that move is called, the contents have already moved, so from should never be dereferenced. If no move routine is supplied (i.e. it is nil), then calling poolcompact is a no-op.

When the pool routines need to allocate a new arena but cannot, either because alloc has returned nil or because doing so would use more than maxsize bytes, poolcompact is called once to defragment the memory and the request is retri.

Pools are protected by the pool routines calling lock (when non-nil) before modifying the pool, and calling unlock when finished.

When internal corruption is detected, panic is called with a print(2) style argument that specifies what happened. It is assumed that panic never returns. When the pool routines wish to convey a message to the caller (usually because logging is turned on; see below), print is called, also with a print(2) style argument.

Flags is a bit vector that tweaks the behavior of the pool routines in various ways. Most are useful for debugging in one way or another. When POOL_ANTAGONISM is set, poolalloc fills blocks with non-zero garbage before releasing them to the user, and poolfree fills the blocks on receipt. This tickles both user programs and the innards of the allocator. Specifically, each 32-bit word of the memory is marked with a pointer value exclusive-or’ed with a constant. The pointer value is the pointer to the beginning of the allocated block and the constant varies in order to distinguish different markings. Freed blocks use the constant 0xF7000000, newly allocated blocks 0xF9000000, and newly created unallocated blocks 0xF1000000. For example, if POOL_ANTAGONISM is set and poolalloc returns a block starting at 0x00012345, each word of the block will contain the value 0xF90012345. Recognizing these numbers in memory-related crashes can help diagnose things like double-frees or dangling pointers.
Setting POOL_PARANOIA causes the allocator to walk the entire pool whenever locking or unlocking itself, looking for corruption. This slows runtime by a few orders of magnitude when many blocks are in use. If POOL_VERBOSITY is set, the entire pool structure is printed (via print) each time the pool is locked or unlocked. POOL_DEBUGGING enables internal debugging output, whose format is unspecified and volatile. It should not be used by most programs. When POOL_LOGGING is set, a single line is printed via print at the beginning and end of each pool call. If logstack is not nil, it will be called as well. This provides a mechanism for external programs to search for leaks. (See leak(1) for one such program.)

The pool routines are strict about the amount of space callers use. If even a single byte is written past the end of the allotted space of a block, they will notice when that block is next used in a call to poolrealloc or free (or at the next entry into the allocator, when POOL_PARANOIA is set), and panic will be called. Since forgetting to allocate space for the terminating NUL on strings is such a common error, if POOL_TOLERANCE is set and a single NUL is found written past the end of a block, print will be called with a notification, but panic will not be.

When POOL_NOREUSE is set, poolfree fills the passed block with garbage rather than return it to the free pool.

**SOURCE**

/sys/src/libc/port/pool.c

**SEE ALSO**

malloc(2), brk(2)

/sys/src/libc/port/malloc.c is a complete example.
NAME
postnote - send a note to a process or process group

SYNOPSIS
#include <u.h>
#include <libc.h>
int postnote(int who, int pid, char *note)

DESCRIPTION
Postnote sends a note to a process or process group. If who is PNPROC, then note is written to /proc/pid/note. If who is PNGROUP, the note is delivered to the process group by writing note to /proc/pid/notepg. For PNGROUP only, if the calling process is in the target group, the note is not delivered to that process.

If the write is successful, zero is returned. Otherwise -1 is returned.

SOURCE
/sys/src/libc/9sys/postnote.c

SEE ALSO
notify(2), intro(2), proc(3)

DIAGNOSTICS
Sets errstr.
NAME
  genprime, gensafeprime, genstrongprime, DSAprimes, probably_prime, smallprimetest - prime number generation

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  #include <mp.h>
  #include <libsec.h>
  int smallprimetest(mpint *p)
  int probably_prime(mpint *p, int nrep)
  void genprime(mpint *p, int n, int nrep)
  void gensafeprime(mpint *p, mpint *alpha, int n, int accuracy)
  void genstrongprime(mpint *p, int n, int nrep)
  void DSAprimes(mpint *q, mpint *p, uchar seed[SHA1dlen])

DESCRIPTION
  Public key algorithms abound in prime numbers. The following routines generate primes or test numbers for primality.

  Smallprimetest checks for divisibility by the first 10000 primes. It returns 0 if p is not divisible by the primes and −1 if it is.

  Probably_prime uses the Miller–Rabin test to test p. It returns non-zero if P is probably prime. The probability of it not being prime is 1/4**nrep.

  Genprime generates a random n bit prime. Since it uses the Miller–Rabin test, nrep is the repetition count passed to probably_prime. Gensafeprime generates an n-bit prime p and a generator alpha of the multiplicative group of integers mod p; there is a prime q such that p−1=2**q. Genstrongprime generates a prime, p, with the following properties:
  - (p−1)/2 is prime. Therefore p−1 has a large prime factor, p'.
  - p'−1 has a large prime factor
  - p+1 has a large prime factor

  DSAprimes generates two primes, q and p, using the NIST recommended algorithm for DSA primes. q divides p−1. The random seed used is also returned, so that skeptics can later confirm the computation. Be patient; this is a slow algorithm.

SOURCE
  /sys/src/libsec

SEE ALSO
  aes(2) blowfish(2), des(2), elgamal(2), rsa(2)
NAME
print, fprintf, sprint, snprintf, seprint, smprint, runesprint, runesnprintf, runeseprint, runesmprintf, vfprintf, vsnprintf, vseprintf, vsmprintf, runevsnprintf, runevseprintf, runevsmprint – print formatted output

SYNOPSIS
#include <u.h>
#include <libc.h>
int print(char *format, ...)
int fprintf(int fd, char *format, ...)
int sprint(char *s, char *format, ...)
int snprintf(char *s, int len, char *format, ...)
char* seprint(char *s, char *e, char *format, ...)
char* smprintf(char *format, ...)
int runesprint(Rune *s, char *format, ...)
int runesnprintf(Rune *s, int len, char *format, ...)
Rune* runeseprintf(Rune *s, Rune *e, char *format, ...)
Rune* runesmprintf(char *format, ...)
int vfprintf(int fd, char *format, va_list v)
int vsnprintf(char *s, int len, char *format, va_list v)
char* vseprintf(char *s, char *e, char *format, va_list v)
char* vsmpprint(char *format, va_list v)
int runevsnprintf(Rune *s, int len, char *format, va_list v)
Rune* runevseprintf(Rune *s, Rune *e, char *format, va_list v)
Rune* runevsmprint(Rune *format, va_list v)

DESCRIPTION
Print writes text to the standard output. Fprint writes to the named output file descriptor; a buffered form is described in bio(2). Sprint places text followed by the NULL character (\0) in consecutive bytes starting at \s; it is the user's responsibility to ensure that enough storage is available. Each function returns the number of bytes transmitted (not including the NULL in the case of sprint), or a negative value if an output error was encountered.

Snprintf is like sprint, but will not place more than len bytes in s. Its result is always NULL-terminated and holds the maximal number of complete UTF-8 characters that can fit. Seprintf is like snprintf, except that the end is indicated by a pointer e rather than a count and the return value points to the terminating NULL of the resulting string. Smprintf is like sprint, except that it prints into and returns a string of the required length, which is allocated by malloc(2).

The routines runesprintf, runesnprintf, runeseprintf, and runesmprintf are the same as sprint, snprintf, seprintf and sprintf except that their output is rune strings instead of byte strings.

Finally, the routines vfprintf, vsnprintf, vseprintf, vsmprintf, runevsnprintf, runevseprintf, and runevsmprint are like their v-less relatives except they take as arguments a va_list parameter, so they can be called within a variadic function. The Example section shows a representative usage.

Each of these functions converts, formats, and prints its trailing arguments under control of a format string. The format contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more arguments. The results are undefined if there are arguments of the wrong type or too few arguments for the format. If the format is exhausted while arguments remain, the excess is ignored.
Each conversion specification has the following format:

    % [flags] verb

The verb is a single character and each flag is a single character or a (decimal) numeric string. Up to two numeric strings may be used: the first is called width, the second precision. A period can be used to separate them, and if the period is present then width and precision are taken to be zero if missing, otherwise they are 'omitted'. Either or both of the numbers may be replaced with the character *, meaning that the actual number will be obtained from the argument list as an integer. The flags and numbers are arguments to the verb described below.

The numeric verbs d, o, b, x, and X format their arguments in decimal, octal, binary, hexadecimal, and upper case hexadecimal. Each interprets the flags 0, h, hh, l, u, +, −, , , and # to mean pad with zeros, short, byte, long, unsigned, always print a sign, left justified, commas every three digits, and alternate format. Also, a space character in the flag position is like +, but prints a space instead of a plus sign for non-negative values. If neither short nor long is specified, then the argument is an int. If unsigned is specified, then the argument is interpreted as a positive number and no sign is output. If two l flags are given, then the argument is interpreted as a vlong (usually an 8-byte, sometimes a 4-byte integer). If precision is not omitted, the number is padded on the left with zeros until at least precision digits appear. Then, if alternate format is specified, for o conversion, the number is preceded by a 0 if it doesn’t already begin with one; for x conversion, the number is preceded by 0x; for X conversion, the number is preceded by 0X. Finally, if width is not omitted, the number is padded on the left (or right, if left justification is specified) with enough blanks to make the field at least width characters long.

The floating point verbs f, e, E, g, and G take a double argument. Each interprets the flags +, −, and # to mean always print a sign, left justified, and alternate format. Width is the minimum field width and, if the converted value takes up less than width characters, it is padded on the left (or right, if 'left justified') with spaces. Precision is the number of digits that are converted after the decimal place for e, E, and f conversions, and precision is the maximum number of significant digits for g and G conversions. The f verb produces output of the form [−]digits[digits]. E conversion appends an exponent E[−]digits, and e conversion appends an exponent e[−]digits. The g verb will output the argument in either e or f with the goal of producing the smallest output. Also, trailing zeros are omitted from the fraction part of the output, and a trailing decimal point appears only if it is followed by a digit. The G verb is similar, but uses E format instead of e. When alternate format is specified, the result will always contain a decimal point, and for g and G conversions, trailing zeros are not removed.

The s verb copies a null-terminated string (pointer to char) to the output. The number of characters copied (n) is the minimum of the size of the string and precision. These n characters are justified within a field of width characters as described above. If a precision is given, it is safe for the string not to be null-terminated as long as it is at least precision characters (not bytes!) long. The S verb is similar, but it interprets its pointer as an array of runes (see utf(6)); the runes are converted to UTF before output.

The c verb copies a single char (promoted to int) justified within a field of width characters as described above. The C verb is similar, but works on runes.

The p verb formats a single pointer or pointer-sized integer (uintptr, see intro(2)) in hexadecimal.

The r verb takes no arguments; it copies the error string returned by a call to strerror(2).

Custom verbs may be installed using fmtinstall(2).

EXAMPLE

This function prints an error message with a variable number of arguments and then quits.

    void fatal(char *msg, ...)
    {
        char buf[1024], *out;
        va_list arg;

        out = seprint(buf, buf+sizeof(buf), "Fatal error: ");
        va_start(arg, msg);
        out = vseprint(out, buf+sizeof(buf), msg, arg);
    }
va_end(arg);
write(2, buf, out-buf);
exits("fatal error");
}

SOURCE
/sys/src/libc/fmt

SEE ALSO
fmtinstall(2), fprintf(2), utf(6), errstr(2)

DIAGNOSTICS
Routines that write to a file descriptor or call malloc set errstr.

BUGS
The formatting is close to that specified for ANSI fprintf(2); the main difference is that b is not in
ANSI and u is a flag here instead of a verb. Also, and distinctly not a bug, print and friends generate
UTF rather than ASCII.

There is no runeprint, runefprint, etc. because runes are byte-order dependent and
should not be written directly to a file; use the UTF output of print or fprintf instead. Also, sprint is
deprecated for safety reasons; use snprint, seprint, or smprint instead. Safety also precludes the
existence of runesprint.
NAME
privalloc, privfree - per-process private storage management

SYNOPSIS
#include <u.h>
#include <libc.h>
void** privalloc(void)
void privfree(void **p)

DESCRIPTION
Privalloc returns a pointer to a per-process private storage location. The location is not shared among processes, even if they share the same data segments. It returns nil if there are no free slots available.

Privfree releases a location allocated with privalloc. It is legal to call privfree with p set to nil.

SOURCE
/sys/src/libc/9sys/privalloc.c

SEE ALSO
exec(2)
NAME
prof - accumulate histogram of process execution

SYNOPSIS
#include <u.h>
#include <libc.h>
void prof(void (*fn)(void*), void *arg, int entries, int what);

DESCRIPTION
Prof arranges to accumulate entries of what data into a histogram by sampling from clock interrupts and recording PCs, calls fn(arg), and finally dumps the accumulated profile. Possible values of what are:
- Profoff: No profiling.
- Profuser: Measure user time only.
- Profkernel: Measure user + kernel time.
- Proftime: Measure total time.
- Profsample: Use clock interrupt to sample.

FILES
prof.out histogram
prof.pid histogram for pid.

SOURCE
/sys/src/libc/port/profile.c

SEE ALSO
prof(1), exec(2)
NAME
   rdproto - parse and process a proto file listing

SYNOPSIS
   #include <u.h>
   #include <libc.h>
   #include <disk.h>
   typedef void Protoenum(char *new, char *old, Dir *d, void *a)
   typedef void Protowarn(char *msg, void *a)
   int rdproto(char *proto, char *root, Protoenum *enm,
                Protowarn *warn, void *a)

DESCRIPTION
   Rdproto reads and interprets the named proto file relative to the root directory root.
   Each line of the proto file specifies a file to copy. Blank lines and lines beginning with #
   are ignored. Indentation (usually tabs) is significant, with each level of indentation corresponding
   to a level in the file tree. Fields within a line are separated by white space. The first field is the last
   path element in the destination file tree. The second field specifies the permissions. The third
   field is the owner of the file, and the fourth is the group owning the file. The fifth field is the name
   of the file from which to copy; this file is read from the current name space, not the source file
   tree. All fields except the first are optional. Specifying – for permissions, owner, or group causes
   rdproto to fetch the corresponding information from the file rather than override it. (This is the
   default behavior when the fields are not present; explicitly specifying – is useful when one wishes
   to set, say, the file owner without setting the permissions.)
   Names beginning with a $ are expanded as environment variables. If the first file specified in a
   directory is *, all of the files in that directory are considered listed. If the first file is +, all of the
   files are copied, and all subdirectories are recursively considered listed. All files are considered
   relative to root.
   For each file named by the proto, enm is called with new pointing at the name of the file (without
   the root prefix), old pointing at the name of the source file (with the root prefix, when applicable),
   and Dir at the desired directory information for the new file. Only the name, uid, gid, mode,
   mtime, and length fields are guaranteed to be valid. The argument a is the same argument
   passed to rdproto; typically it points at some extra state used by the enumeration function.
   When files or directories do not exist or cannot be read by rdproto, it formats a warning message,
   calls warn, and continues processing; if warn is nil, rdproto prints the warning message to stand-
   ard error.
   Rdproto returns zero if proto was processed, -1 if it could not be opened.

FILES
   /sys/lib/sysconfig/proto/ directory of prototype files.
   /sys/lib/sysconfig/proto/portproto generic prototype file.

SOURCE
   /sys/src/libdisk/proto.c

SEE ALSO
   mk9660(8), mkfs(8)
NAME
pushssl - attach SSL version 2 encryption to a communication channel

SYNOPSIS
#include <u.h>
#include <libc.h>
int pushssl(int fd, char *alg, char *secin, char *secout, int *cfd)

DESCRIPTION
Pushssl opens an ssl(3) device, connects it to the communications channel \texttt{fd}, and starts up encryption and message authentication as specified in \texttt{alg}. The algorithms are separated by a space and either can be first. See ssl(3) for the possible algorithms. \texttt{Secin} and \texttt{secout} contain the encryption keys for the two directions. If either is nil, the other is used in both directions. If \texttt{cfd} is non-nil, the SSL control channel is opened and its \texttt{fd} returned.

Pushssl returns a file descriptor for the SSL data channel. Anything written to this descriptor will get encrypted and authenticated and then written to the file descriptor, \texttt{fd}.

SOURCE
/sys/src/libc/9sys

SEE ALSO
dial(2), ssl(3),

DIAGNOSTICS
return –1 on failure.
NAME
pushtls, tlsClient, tlsServer, initThumbprints, freeThumbprints, okThumbprint, readcert, readcertchain - attach TLS1 or SSL3 encryption to a communication channel

SYNOPSIS
#include <u.h>
#include <libsec.h>

int pushtls(int fd, char *hashalg, char *encalg,
    int isclient, char *secret, char *dir)
#include <mp.h>
#include <libsec.h>

int tlsClient(int fd, TLSconn *conn)
int tlsServer(int fd, TLSconn *conn)
uchar *readcert(char *filename, int *pcertlen)
PEMChain *readcertchain(char *filename)
Thumbprint *initThumbprints(char *ok, char *crl)
void freeThumbprints(Thumbprint *table)

int okThumbprint(uchar *hash, Thumbprint *table)

DESCRIPTION
Transport Layer Security (TLS) comprises a record layer protocol, doing message digesting and encrypting in the kernel, and a handshake protocol, doing initial authentication and secret creation at user level and then starting a data channel in the record protocol. TLS is nearly the same as SSL 3.0, and the software should interoperable with implementations of either standard.

To use just the record layer, as described in tls(3), call pushtls to open the record layer device, connect to the communications channel fd, and start up encryption and message authentication as specified in hashalg, encalg, and secret. These parameters must have been arranged at the two ends of the conversation by other means. For example, hashalg could be sha1, encalg could be rc4_128, and secret could be the base-64 encoding of two (client-to-server and server-to-client) 20-byte digest keys and two corresponding 16-byte encryption keys. Pushtls returns a file descriptor for the TLS data channel. Anything written to this descriptor will get encrypted and authenticated and then written to the file descriptor, fd. If dir is non-zero, the path name of the connection directory is copied into dir. This path name is guaranteed to be less than 40 bytes long.

Certificates
Alternatively, call tlsClient to speak the full handshake protocol, negotiate the algorithms and secrets, and return a new data file descriptor for the data channel. Conn points to a (caller-allocated) struct:

typedef struct TLSconn {
    char dir[40];  /* OUT  connection directory */
    uchar *cert;  /* IN/OUT certificate */
    uchar *sessionID;  /* IN/OUT session ID */
    int certlen, sessionIDlen;
    void (*trace)(char*fmt, ...);
    PEMChain *chain;
    char *sessionType;  /* opt IN  session type */
    uchar *sessionKey;  /* opt IN/OUT session key */
    int sessionKeylen;  /* opt IN session key length */
    char *sessionConst;  /* opt IN session constant */
} TLSconn;

defined in tls.h. On input, the caller can provide options such as cert, the local certificate, and sessionID, used by a client to resume a previously negotiated security association. On output, the connection directory is set, as with listen (see dial(2)). The input cert is freed and a freshly allocated copy of the remote’s certificate is returned in conn, to be checked by the caller according to
its needs. One way to check the remote certificate is to use initThumbprints and freeThumbprints which allocate and free, respectively, a table of hashes from files of known trusted and revoked certificates. okThumbprint confirms that a particular hash is in the table.

TlsClient will optionally compute a session key for use by higher-level protocols. To compute a session key, the caller must set sessionType to a known session type; sessionKeylen to the desired key length; sessionKey to a buffer of length sessionKeylen; and sessionConst to the desired salting constant. The only supported session type is tls, as used by 802.1x.

TlsServer executes the server side of the handshake. The caller must initialize conn->cert, usually by calling readcert to read and decode the PEM-encoded certificate from filename, return a pointer to malloced storage containing the certificate, and store its length through pcertlen. The private key corresponding to cert.pem should have been previously loaded into factotum. (See rsa(8) for more about key generation.)

Readcertchain will read a PEM-encoded chain of certificates from filename and return a pointer to a linked list of malloced PEMChain structures, defined in tls.h:

```c
typedef struct PEMChain PEMChain;
struct PEMChain {
    PEMChain *next;
    uchar *pem;
    int pemlen;
};
```

By setting

```c
conn->chain = readcertchain("intermediate-certs.pem");
```

the server can present extra certificate evidence to establish the chain of trust to a root authority known to the client.

Conn is not required for the ongoing conversation and may be freed by the application whenever convenient.

EXAMPLES

Start the client half of TLS and check the remote certificate:

```c
uchar hash[SHA1dlen];

conn = (TLSconn*)malloc(sizeof *conn, 1);
fd = tlsClient(fd, conn);
shal(conn->cert, conn->certlen, hash, nil);
if(!okThumbprint(hash,table))
    exits("suspect server");
```

Run the server side:

```c
fd = accept(lcfd, ldir);
conn = (TLSconn*)malloc(sizeof *conn, 1);
conn->cert = readcert("cert.pem", &conn->certlen);
fd = tlsServer(fd, conn);
```

FILES

/sys/lib/tls  thumbprints of trusted services
/sys/lib/ssl  PEM certificate files

SOURCE

/sys/src/libc/9sys/pushtls.c
/sys/src/libsec/port

SEE ALSO
dial(2), tls(3), factotum(4), thumbprint(6)

DIAGNOSTICS

Return -1 on failure.

BUGS

Client certificates and client sessionIDs are not yet implemented.
Note that in the TLS protocol `sessionID` itself is public; it is used as a pointer to secrets stored in `factotum`. 
NAME
qball - 3-d rotation controller

SYNOPSIS
#include <draw.h>
#include <geometry.h>

void qball(Rectangle r, Mouse *mousep,
            Quaternion *orientation,
            void (*redraw)(void), Quaternion *ap)

DESCRIPTION
Qball is an interactive controller that allows arbitrary 3-space rotations to be specified with the mouse. Imagine a sphere with its center at the midpoint of rectangle r, and diameter the smaller of r’s dimensions. Dragging from one point on the sphere to another specifies the endpoints of a great-circle arc. (Mouse points outside the sphere are projected to the nearest point on the sphere.) The axis of rotation is normal to the plane of the arc, and the angle of rotation is twice the angle of the arc.

Argument mousep is a pointer to the mouse event that triggered the interaction. It should have some button set. Qball will read more events into mousep, and return when no buttons are down.

While qball is reading mouse events, it calls out to the caller–supplied routine redraw, which is expected to update the screen to reflect the changing orientation. Argument orientation is the orientation that redraw should examine, represented as a unit Quaternion (see quaternion(9.2)). The caller may set it to any orientation. It will be updated before each call to redraw (and on return) by multiplying by the rotation specified with the mouse.

It is possible to restrict qball’s attention to rotations about a particular axis. If ap is null, the rotation is unconstrained. Otherwise, the rotation will be about the same axis as *ap. This is accomplished by projecting points on the sphere to the nearest point also on the plane through the sphere’s center and normal to the axis.

SOURCE
/sys/src/libgeometry/qball.c

SEE ALSO

quaternion(2)
NAME
qsort - quicker sort

SYNOPSIS
#include <u.h>
#include <libc.h>
void qsort(void *base, long nel, long width,
    int (*compar)(void*, void*))

DESCRIPTION
Qsort (quicker sort) sorts an array into nondecreasing order. The first argument is a pointer to the
base of the data; the second is the number of elements; the third is the width of an element in
bytes; the last is the name of a comparison routine to be called with pointers to elements being
compared. The routine must return an integer less than, equal to, or greater than 0 according as
the first argument is to be considered less than, equal to, or greater than the second.

SOURCE
/sys/src/libc/port/qsort.c

SEE ALSO
sort(1)
NAME
   qtom, mtoq, qadd, qsub, qneg, qmul, qdiv, qunit, qinv, qlen, slerp, qmid, qsqrt - Quaternion arithmetic

SYNOPSIS
   #include <draw.h>
   #include <geometry.h>
   Quaternion qadd(Quaternion q, Quaternion r)
   Quaternion qsub(Quaternion q, Quaternion r)
   Quaternion qneg(Quaternion q)
   Quaternion qmul(Quaternion q, Quaternion r)
   Quaternion qdiv(Quaternion q, Quaternion r)
   Quaternion qinv(Quaternion q)
   double qlen(Quaternion p)
   Quaternion qunit(Quaternion q)
   void qtom(Matrix m, Quaternion q)
   Quaternion mtoq(Matrix mat)
   Quaternion slerp(Quaternion q, Quaternion r, double a)
   Quaternion qmid(Quaternion q, Quaternion r)
   Quaternion qsqrt(Quaternion q)

DESCRIPTION
   The Quaternions are a non-commutative extension field of the Real numbers, designed to do for
   rotations in 3-space what the complex numbers do for rotations in 2-space. Quaternions have a
   real component r and an imaginary vector component v=(i,j,k). Quaternions add componentwise and
   multiply according to the rule \((r,v)(s,w) = (rs-v\cdot w, rw+vs+w\times v)\), where \(\cdot\) and \(\times\)
   are the ordinary vector dot and cross products. The multiplicative inverse of a non-zero quaternion
   \((r,v)\) is \((r,-v)/(r^2-v\cdot v)\).
   
   The following routines do arithmetic on quaternions, represented as

   typedef struct Quaternion Quaternion;
   struct Quaternion{
       double r, i, j, k;
   };

Name  Description
qadd  Add two quaternions.
qsub  Subtract two quaternions.
qneg  Negate a quaternion.
qmul  Multiply two quaternions.
qdiv  Divide two quaternions.
qinv  Return the multiplicative inverse of a quaternion.
qlen  Return \(\sqrt{q.r^2+q.i^2+q.j^2+q.k^2}\), the length of a quaternion.
qunit Return a unit quaternion \((length=1)\) with components proportional to \(q\)’s.

A rotation by angle \(\theta\) about axis \(A\) (where \(A\) is a unit vector) can be represented by the unit
quaternion \(q=(\cos \theta/2, \sin \theta/2 A)\). The same rotation is represented by \(-q\); a rotation by \(-\theta\)
about \(-A\) is the same as a rotation by \(\theta\) about \(A\). The quaternion \(q\) transforms points by \((0,x',y',z') = q^{-1}(0,x,y,z)q\). Quaternion
multiplication composes rotations. The orientation of an object in 3-space can be represented by a quaternion giving its rotation relative to some ‘standard’ orientation.

The following routines operate on rotations or orientations represented as unit quaternions:
mtoq  Convert a rotation matrix (see \textit{matrix(2)}) to a unit quaternion.
qtom  Convert a unit quaternion to a rotation matrix.
slerp  Spherical lerp. Interpolate between two orientations. The rotation that carries q to r is $q^{-1}r$, so slerp(q, r, t) is $q(q^{-1}r)t$.
qmid  slerp(q, r, .5)
qsqrt  The square root of q. This is just a rotation about the same axis by half the angle.

SOURCE
/sys/src/libgeometry/quaternion.c

SEE ALSO
  matrix(2), qball(2)
NAME
quotestrdup, quoterunestrdup, unquotestrdup, unquoterunestrdup, quotestrfmt, quoterunestrfmt,
quotefmtinstall, doquote, needsrcquote – quoted character strings

SYNOPSIS
#include <u.h>
#include <libc.h>
char *quotestrdup(char *s)
Rune *quoterunestrdup(Rune *s)
char *unquotestrdup(char *s)
Rune *unquoterunestrdup(Rune *s)
int quotestrfmt(Fmt*)
int quoterunestrfmt(Fmt*)
void quotefmtinstall(void)
int (*doquote)(int c)
int needsrcquote(int c)

DESCRIPTION
These routines manipulate character strings, either adding or removing quotes as necessary. In
the quoted form, the strings are in the style of rc(1), with single quotes surrounding the string.
Embedded single quotes are indicated by a doubled single quote. For instance,

'Don’t worry!
when quoted becomes

'Don’t worry!

The empty string is represented by two quotes, ‘’.

The first four functions act as variants of strdup (see strcat(2)). Each returns a freshly allocated
copy of the string, created using malloc(2). Quotestrdup returns a quoted copy of s, while
unquotestrdup returns a copy of s with the quotes evaluated. The rune versions of these functions
do the same for strings (see runestrcat(2)).

The string returned by quotestrdup or quoterunestrdup has the following properties:
1. If the original string s is empty, the returned string is ‘’.
2. If s contains no quotes, blanks, or control characters, the returned string is identical to s.
3. If s needs quotes to be added, the first character of the returned string will be a quote. For
   example, hello world becomes 'hello world' not hello 'world.'

The function pointer doquote is nil by default. If it is non-nil, characters are passed to that function
to see if they should be quoted. This mechanism allows programs to specify that characters
other than blanks, control characters, or quotes be quoted. Regardless of the return value of
*doquote, blanks, control characters, and quotes are always quoted. Needsrcquote is provided as
a doquote function that flags any character special to rc(1).

Quotestrfmt and quoterunestrfmt are print(2) formatting routines that produce quoted strings as
output. They may be installed by hand, but quotefmtinstall installs them under the standard for-
mat characters q and Q. (They are not installed automatically.) If the format string includes the
alternate format character #, for example %#q, the printed string will always be quoted; otherwise
quotes will only be provided if necessary to avoid ambiguity. In <libc.h> there are #pragma
statements so the compiler can type-check uses of %q and %Q in print(2) format strings.

SOURCE
/sys/src/libc/port/quote.c
/sys/src/libc/fmt/fmtquote.c

SEE ALSO
rc(1), malloc(2), print(2), strcat(2)
NAME
rand, lrand, frand, nrand, lnrand, srand, truerand, ntruerand, genrandom, prng, fastrand, nfastrand – random number generators

SYNOPSIS
#include <u.h>
#include <libc.h>
int    rand(void)
long   lrand(void)
double frand(void)
int    nrand(int val)
long   lnrand(long val)
void   srand(long seed)
ulong  truerand(void)
ulong  ntruerand(ulong val)

#include <mp.h>
#include <libsec.h>
void   genrandom(uchar *buf, int nbytes)
void   prng(uchar *buf, int nbytes)
ulong  fastrand(void)
ulong  nfastrand(ulong val)

DESCRIPTION
Rand returns a uniform pseudo-random number x, 0\leq x<2^{15}.

Lrand returns a uniform long x, 0\leq x<2^{31}.

Frand returns a uniform double x, 0.0\leq x<1.0. This function calls lrand twice to generate a number with as many as 62 significant bits of mantissa.

Nrand returns a uniform integer x, 0\leq x<val. Lnrand is the same, but returns a long.

The algorithm is additive feedback with:
\[ x[n] = (x[n-273] + x[n-607]) \mod 2^{31} \]
giving a period of \(2^{30} \times (2^{607} - 1)\).

The generators are initialized by calling srand with whatever you like as argument. To get a different starting value each time,
srand(time(0))
will work as long as it is not called more often than once per second. Calling
srand(1)
will initialize the generators to their starting state.

Truerand returns a random unsigned long read from /dev/random. Due to the nature of /dev/random, truerand can only return a few hundred bits a second.

Ntruerand returns a uniform random integer x, 0\leq x<val\leq2^{32}-1.

Genrandom fills a buffer with bytes from the X9.17 pseudo–random number generator. The X9.17 generator is seeded by 24 truly random bytes read from /dev/random.

Prng uses the native rand(2) pseudo–random number generator to fill the buffer. Used with srand, this function can produce a reproducible stream of pseudo random numbers useful in testing.

Both genrandom and prng may be passed to mprand (see mp(2)).
Fastrand uses genrandom to return a uniform unsigned long \( x, 0 \leq x < 2^{32} - 1 \).

Nfastrand uses genrandom to return a uniform unsigned long \( x, 0 \leq x < \text{val} \leq 2^{32} - 1 \).

**SOURCE**
/sys/src/libc/port/*rand.c
/sys/src/libc/9sys/truerand.c
/sys/src/libsec/port/genrandom.c
/sys/src/libsec/port/prng.c
/sys/src/libsec/port/*fastrand.c

**SEE ALSO**
cons(3), mp(2)

**BUGS**
Truerand and ntruerand maintain a static file descriptor.
NAME
setupRC4state, rc4, rc4skip, rc4back – alleged rc4 encryption

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <mp.h>
#include <libsec.h>
void setupRC4state(RC4state *s, uchar *seed, int slen)
void rc4(RC4state *s, uchar *data, int dlen)
void rc4skip(RC4state *s, int nbytes)
void rc4back(RC4state *s, int nbytes)

DESCRIPTION
This is an algorithm alleged to be Rivest’s RC4 encryption function. It is a pseudo-random number generator with a 256 byte state and a long cycle. The input buffer is XOR’d with the output of the generator both to encrypt and to decrypt. The seed, entered using setupRC4state, can be any length. The generator can be run forward using rc4, skip over bytes using rc4skip to account lost transmissions, or run backwards using rc4back to cover retransmitted data. The RC4state structure keeps track of the algorithm.

SOURCE
/sys/src/libsec

SEE ALSO
mp(2), aes(2), blowfish(2), des(2), dsa(2), elgamal(2), rsa(2), sechash(2), prime(2), rand(2)
NAME
read, readn, write, pread, pwrite – read or write file

SYNOPSIS
#include <u.h>
#include <libc.h>
long read(int fd, void *buf, long nbytes)
long readn(int fd, void *buf, long nbytes)
long write(int fd, void *buf, long nbytes)
long pread(int fd, void *buf, long nbytes, vlong offset)
long pwrite(int fd, void *buf, long nbytes, vlong offset)

DESCRIPTION
Read reads nbytes bytes of data from the offset in the file associated with fd into memory at buf. The offset is advanced by the number of bytes read. It is not guaranteed that all nbytes bytes will be read; for example if the file refers to the console, at most one line will be returned. In any event the number of bytes read is returned. A return value of 0 is conventionally interpreted as end of file.

Readn is just like read, but does successive read calls until nbytes have been read, or a read system call returns a non-positive count.

Write writes nbytes bytes of data starting at buf to the file associated with fd at the file offset. The offset is advanced by the number of bytes written. The number of characters actually written is returned. It should be regarded as an error if this is not the same as requested.

Pread and Pwrite are equivalent to a seek(2) to offset followed by a read or write. By combining the operations in a single atomic call, they more closely match the 9P protocol (see intro(5)) and, more important, permit multiprocess programs to execute multiple concurrent read and write operations on the same file descriptor without interference.

SOURCE
/sys/src/libc/9syscall
/sys/src/libc/port/readn.c

SEE ALSO
intro(2), dirread(2), dup(2), open(2), pipe(2), readv(2)

DIAGNOSTICS
These functions set errstr.
NAME
   RGB, readcolmap, writecolmap - access display color map

SYNOPSIS
   #include <u.h>
   #include <libc.h>
   #include <draw.h>
   int readcolmap(Display *d, RGB *map)
   int writecolmap(Display *d, RGB *map)

DESCRIPTION
   Colors are described by their red, green, and blue light intensities, in an RGB datum:

   typedef
   struct RGB {
      ulong red;
      ulong green;
      ulong blue;
   } RGB;

   Black is represented by zero in all three positions and white has the maximum unsigned long value in all three positions.

   A color map is an array of RGBs, of length $2^{\text{depth}}$, giving the colors for pixels 0, 1, 2, etc. On displays with color mapped pixels (typically 8-bit displays), one retrieves RGB color information by treating the pixel data as an offset into the color map.

   Readcolmap reads the color map for the given display into the provided map, which must have enough space to hold it. Writecolmap associates the given color map with the given display, if possible. (The hardware might not allow this.) Both return 0 on success, or -1 on error, setting errstr.

   Changing the hardware color map does not change the color map used by the draw(2) operator to convert between mapped and true color or greyscale images, which is described in color(6).

SOURCE
   /sys/src/libdraw

SEE ALSO
   graphics(2), draw(3), color(6)
NAME
readv, writev, preadv, pwrites - scatter/gather read and write

SYNOPSIS
#include <u.h>
#include <libc.h>
typedef
struct IOchunk
{
  void *addr;
  ulong len;
} IOchunk;
long readv(int fd, IOchunk *io, int nio)
long preadv(int fd, IOchunk *io, int nio, vlong off)
long writev(int fd, IOchunk *io, int nio)
long pwritev(int fd, IOchunk *io, int nio, vlong off)

DESCRIPTION
These functions supplement the standard read and write operations of read(2) with facilities for scatter/gather I/O. The set of I/O buffers is collected into an array of IOchunk structures passed as an argument.

readv reads data from fd and returns the total number of bytes received. The received data is stored in the successive nio elements of the IOchunk array, storing io[0].len bytes at io[0].addr, the next io[1].len at io[1].addr, and so on. preadv does the same, but implicitly seeks to I/O offset off by analogy with readv.

Writev and pwritev are the analogous write routines.

SOURCE
/sys/src/libc/9sys/readv.c
/sys/src/libc/9sys/writev.c

SEE ALSO
intro(2), read(2)

DIAGNOSTICS
These functions set errstr.

BUGS
The implementations use malloc(2) to build a single buffer for a standard call to read or write. They are placeholders for possible future system calls.
NAME
regcomp, regcomplit, regcompnl, regexec, regsub, regexec, regsub, regerror - regular expression

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <regexp.h>
Reprog *regcomp(char *exp)
Reprog *regcomplit(char *exp)
Reprog *regcompnl(char *exp)
int regexec(Reprog *prog, char *string, Resub *match, int msize)
void regsub(char *source, char *dest, int dlen, Resub *match, int msize)
int rregexec(Reprog *prog, Rune *string, Resub *match, int msize)
void rregsub(Rune *source, Rune *dest, int dlen, Resub *match, int msize)
void regerror(char *msg)

DESCRIPTION
Regcomp compiles a regular expression and returns a pointer to the generated description. The space is allocated by malloc(2) and may be released by free. Regular expressions are exactly as in regexp(3).
Regcomplit is like regcomp except that all characters are treated literally. Regcompnl is like regcomp except that the . metacharacter matches all characters, including newlines.
Regexec matches a null-terminated string against the compiled regular expression in prog. If it matches, regexec returns 1 and fills in the array match with character pointers to the substrings of string that correspond to the parenthesized subexpressions of exp: match[i].sp points to the beginning and match[i].ep points just beyond the end of the ith substring. (Subexpression i begins at the ith left parenthesis, counting from 1.) Pointers in match[0] pick out the substring that corresponds to the whole regular expression. Unused elements of match are filled with zeros. Matches involving *, +, and ? are extended as far as possible. The number of array elements in match is given by msize. The structure of elements of match is:

typedef struct {
    union {
        char *sp;
        Rune *rsp;
    };
    union {
        char *ep;
        Rune *rep;
    };
} Resub;

If match[0].sp is nonzero on entry, regexec starts matching at that point within string. If match[0].ep is nonzero on entry, the last character matched is the one preceding that point.
Regsub places in dest a substitution instance of source in the context of the last regexec performed using match. Each instance of \n, where n is a digit, is replaced by the string delimited by match[n].sp and match[n].ep. Each instance of & is replaced by the string delimited by match[0].sp and match[0].ep. The substitution will always be null terminated and trimmed to fit into dlen bytes.
Regerror, called whenever an error is detected in regcomp, writes the string msg on the standard error file and exits. Regerror can be replaced to perform special error processing. If the user supplied regerror returns rather than exits, regcomp will return 0.
Regexec and rregsub are variants of regexec and regsub that use strings of Runes instead of strings of chars. With these routines, the rsp and rep fields of the match array elements should
be used.

SOURCE
/sys/src/libregexp

SEE ALSO
grep(1)

DIAGNOSTICS
Regcomp returns 0 for an illegal expression or other failure. Regexec returns 0 if string is not matched.

BUGS
There is no way to specify or match a NUL character; NULs terminate patterns and strings.
NAME
remove - remove a file

SYNOPSIS
#include <u.h>
#include <libc.h>
int remove(char *file)

DESCRIPTION
Remove removes file from the directory containing it and discards the contents of the file. The
user must have write permission in the containing directory. If file is a directory, it must be empty.

SOURCE
/sys/src/libc/9syscall

SEE ALSO
intro(2), remove(5), the description of ORCLOSE in open(2).

DIAGNOSTICS
Sets errstr.
NAME
rendezvous - user level process synchronization

SYNOPSIS
#include <u.h>
#include <libc.h>
void* rendezvous(void* tag, void* value)

DESCRIPTION
The rendezvous system call allows two processes to synchronize and exchange a value. In conjunction with the shared memory system calls (see segattach(2) and fork(2)), it enables parallel programs to control their scheduling.

Two processes wishing to synchronize call rendezvous with a common tag, typically an address in memory they share. One process will arrive at the rendezvous first; it suspends execution until a second arrives. When a second process meets the rendezvous the value arguments are exchanged between the processes and returned as the result of the respective rendezvous system calls. Both processes are awakened when the rendezvous succeeds.

The set of tag values which two processes may use to rendezvous—their tag space—is inherited when a process forks, unless RFREND is set in the argument to fork; see fork(2).

If a rendezvous is interrupted the return value is −0, so that value should not be used in normal communication.

SOURCE
/sys/src/libc/9syscall

SEE ALSO
fork(2), lock(2), segattach(2)

DIAGNOSTICS
Sets errstr.
NAME
asn1dump, asn1toRSApriv, decodePEM, rsadecrypt, rsaencrypt, rsagen, rsaprilloc, rsaprivfree, rsapriltopub, rsapuballoc, rsapubfree, X509toRSApub, X509gen, X509verify – RSA encryption algorithm

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <mp.h>
#include <libsec.h>

RSApriv* rsagen(int nlen, int elen, int nrep)
mpint* rsaencrypt(RSApub *k, mpint *in, mpint *out)
mpint* rsadecrypt(RSApriv *k, mpint *in, mpint *out)
RSApub* rsapuballoc(void)
void rsapubfree(RSApub*)
RSApriv* rsaprivalloc(void)
void rsaprivfree(RSApriv*)
RSApub* X509toRSApub(uchar *cert, int ncert, char *name, int nname)
RSApriv* asn1toRSApriv(uchar *priv, int npriv)
void asn1dump(uchar *der, int len)
uchar* decodePEM(char *s, char *type, int *len, char **new_s)
uchar* X509gen(RSApriv *priv, char *subj, ulong valid[2], int *certlen);
uchar* X509req(RSApriv *priv, char *subj, int *certlen);
char* X509verify(uchar *cert, int ncert, RSApub *pk)

DESCRIPTION
RSA is a public key encryption algorithm. The owner of a key publishes the public part of the key:

struct RSApub {
    mpint *n; /* modulus */
    mpint *ek; /* exp (encryption key) */
};

This part can be used for encrypting data (with rsaencrypt) to be sent to the owner. The owner decrypts (with rsadecrypt) using his private key:

struct RSApriv {
    RSApub pub;
    mpint *dk; /* exp (decryption key) */

    /* precomputed crt values */
    mpint *p;
    mpint *q;
    mpint *kp; /* k mod p-1 */
    mpint *kq; /* k mod q-1 */
    mpint *c2; /* for converting residues to number */
};

Keys are generated using rsagen. Rsagen takes both bit length of the modulus, the bit length of the public key exponent, and the number of repetitions of the Miller–Rabin primality test to run. If the latter is 0, it does the default number of rounds. Rsagen returns a newly allocated structure.
containing both public and private keys. Rsaprivtopub returns a newly allocated copy of the public key corresponding to the private key.

The routines rsaalloc, rsafree, rsapuballoc, rsapubfree, rsaprivalloc, and rsaprvfree are provided to aid in user provided key I/O.

Given a binary X.509 cert, the routine X509toRSApub returns the public key and, if name is not nil, the CN part of the Distinguished Name of the certificate's Subject. (This is conventionally a userid or a host DNS name.) No verification is done of the certificate signature; the caller should check the fingerprint, shal(cert), against a table or check the certificate by other means. X.509 certificates are often stored in PEM format; use dec64 to convert to binary before computing the fingerprint or calling X509toRSApub. For the special case of certificates signed by a known trusted key (in a single step, without certificate chains), X509verify checks the signature on cert. It returns nil if successful, else an error string.

X509gen creates a self-signed X.509 certificate, given an RSA keypair priv, a issuer/subject string subj, and the starting and ending validity dates, valid. Length of the allocated binary certificate is stored in certlen. The subject line is conventionally of the form

    C=US ST=NJ L=07922 O=Lucent OU='Bell Labs' CN=Eric

using the quoting conventions of tokenize in getfields(2).

Asn1toRSApriv converts an ASN1 formatted RSA private key into the corresponding RSApriv structure.

Asn1dump prints an ASN1 object to standard output.

DecodePEM takes a zero terminated string, s, and decodes the PEM (privacy-enhanced mail) formatted section for type within it. If successful, it returns malloced storage containing the decoded section, which the caller must free, and sets *len to its decoded length. Otherwise nil is returned and *len is undefined. If not nil, new_s is set to the first character beyond the type section.

SOURCE
/sys/src/libsec

SEE ALSO
mp(2), aes(2), blowfish(2), des(2), dsa(2), elgamal(2), rc4(2), sechash(2), prime(2), rand(2), rsa(8)
NAME
  runetochar, chartorune, runelen, runenlen, fullrune, utfecpy, utflen, utfnlen, utfrune, utfrrune,
  utfutf - rune/UTF conversion

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  int   runetochar(char *, Rune *)
  int   chartorune(Rune *, char *)
  int   runelen(long r)
  int   runenlen(Rune *, int n)
  char* utfecpy(char *, char *, char *)
  int   utflen(char *)
  int   utfnlen(char *, long n)
  char* utfrune(char *, long c)
  char* utfrrune(char *, long c)
  char* utfutf(char *, char *)

DESCRIPTION
  These routines convert to and from a UTF byte stream and runes.

  Runetochar copies one rune at r to at most UTFmax bytes starting at s and returns the number of
  bytes copied. UTFmax, defined as 3 in <libc.h>, is the maximum number of bytes required to
  represent a rune.

  Chartorune copies at most UTFmax bytes starting at s to one rune at r and returns the number of
  bytes copied. If the input is not exactly in UTF format, chartorune will convert to Runeerror
  (0xFFFFD) and return 1.

  Runelen returns the number of bytes required to convert r into UTF.

  Runenlen returns the number of bytes required to convert the n runes pointed to by r into UTF.

  Fullrune returns 1 if the string s of length n is long enough to be decoded by chartorune and 0
  otherwise. This does not guarantee that the string contains a legal UTF encoding. This routine is
  used by programs that obtain input a byte at a time and need to know when a full rune has
  arrived.

  The following routines are analogous to the corresponding string routines with utf substituted
  for str and rune substituted for chr.

  Utfecpy copies UTF sequences until a null sequence has been copied, but writes no sequences
  beyond es1. If any sequences are copied, s1 is terminated by a null sequence, and a pointer to that
  sequence is returned. Otherwise, the original s1 is returned.

  Utflen returns the number of runes that are represented by the UTF string s.

  Utfnlen returns the number of complete runes that are represented by the first n bytes of UTF
  string s. If the last few bytes of the string contain an incompletely coded rune, utfnlen will not
  count them; in this way, it differs from utflen, which includes every byte of the string.

  Utfrune (utfrrune) returns a pointer to the first (last) occurrence of rune c in the UTF string s, or 0
  if c does not occur in the string. The NUL byte terminating a string is considered to be part of the
  string s.

  Utfutf returns a pointer to the first occurrence of the UTF string s2 as a UTF substring of s1, or 0 if
  there is none. If s2 is the null string, utfutf returns s1.

SOURCE
  /sys/src/libc/port/rune.c
  /sys/src/libc/port/utfrune.c
SEE ALSO
   utf(6), tcs(1)
NAME
runestrcat, runestrcat, runestrcmp, runestrcmp, runestrcpy, runestrcpy, runestr-
trlen, runestchr, runestchr, runestrdup, runestrstr - rune string operations

SYNOPSIS
#include <u.h>
#include <libc.h>
Rune* runestrcat(Rune *s1, Rune *s2)
Rune* runestrcat(Rune *s1, Rune *s2, long n)
int runestrcmp(Rune *s1, Rune *s2)
int runestrcmp(Rune *s1, Rune *s2, long n)
Rune* runestrcpy(Rune *s1, Rune *s2)
Rune* runestrcpy(Rune *s1, Rune *s2, long n)
Rune* runestrcpy(Rune *s1, Rune *es1, Rune *s2)
long runestrlen(Rune *s)
Rune* runestchr(Rune *s, Rune c)
Rune* runestchr(Rune *s, Rune c)
Rune* runestdup(Rune *s)
Rune* runestrstr(Rune *s1, Rune *s2)

DESCRIPTION
These functions are rune string analogues of the corresponding functions in strcat(2).

SOURCE
/sys/src/libc/port

SEE ALSO
memory(2), rune(2), strcat(2)

BUGS
The outcome of overlapping moves varies among implementations.
NAME
scribblealloc, recognize - character recognition

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
#include <scribble.h>

Scribble *scribblealloc(void);
Rune recognize(Scribble *);

DESCRIPTION
The scribble library implements simple character recognition. All characters are drawn using a single stroke of the pen (mouse button 1) as on a palmtop computer. A reference card is in /sys/src/libscribble/quickref.gif.

The library is not really intended for standalone use. Its primary use is by the scribble graphical control (see control(2)).

Scribblealloc allocates and returns an appropriately initialized Scribble structure:
#define CS_LETTERS 0
#define CS_DIGITS 1
#define CS_PUNCTUATION 2

struct Scribble {
  /* private state */
  Point *pt;
  int ppsasize;
  Stroke ps;
  Graffiti *graf;
  int capsLock;
  int puncShift;
  int tmpShift;
  int ctrlShift;
  int curCharSet;
};

This structure encodes the points making up the stroke to be recognized, as well as the character group in which the stroke should be searched.

There are three such groups: letters, digits, and punctuation. The current group is encoded in the curCharSet field of the Scribble structure. Special strokes are recognized to switch between groups. In addition, the character recognized is influenced by mode parameters and modifies them. These are identified by the capsLock, puncShift, tmpShift, and ctrlShift fields of the Scribble structure. When puncShift is non-zero, the character is recognized in the punctuation character set. Similarly, when the character recognized is printable and ctrlShift is set, the associated control character is returned as if the control key were depressed, and when the character is a letter and capsLock or tmpShift is set, the uppercase version is returned. The puncShift and tmpShift flags are turned off once a character has been recognized; the others are left set.

The character to be recognized is encoded as an array of pen_points in the ps field. To allow easy drawing of the stroke as it is drawn, the pt and ppsasize fields are available to the application code for storing an array of points for a call to poly (see draw(2)).

Recognize recognizes the character provided in the ps field of the Scribble structure; it returns the rune or zero if nothing was recognized.

FILES
/sys/src/libscribble/quickref.gif serves as a quick reference card.
/sys/lib/scribble/classifiers contains the stroke definitions.
SOURCE
/sys/src/libscribble

This library is adapted from software reproduced by permission:
Graffiti.c is based on the file Scribble.c copyrighted by Keith Packard:

Copyright © 1999 Keith Packard

Permission to use, copy, modify, distribute, and sell this software and its documentation for any
purpose is hereby granted without fee, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission notice appear in supporting docu-
mentation, and that the name of Keith Packard not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission. Keith Packard makes no
representations about the suitability of this software for any purpose. It is provided "as is" without
express or implied warranty.

Portions of the software Copyright © 1994 by Sun Microsystems Computer Company.

Portions of the software Copyright © 2000 by Compaq Computer Corporation.

SEE ALSO
Keyboard and prompter in bitsyload(1), draw(2), control(2)
NAME
openscsi, closescsi, scsiready, scsi, scsicmd, scsierror – SCSI device operations

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <disk.h>

typedef struct Scsi {
    char *inquire;
    int rawfd;
    int nchange;
    ulong changetime;
};
Scsi* openscsi(char *devdir)
void closescsi(Scsi *s)
int scsiready(Scsi *s)
int scsi(Scsi *s, uchar *cmd, int ncmd,
        void *data, int ndata, int dir)
int scsicmd(Scsi *s, uchar *cmd, int ncmd,
        void *data, int ndata, int dir)
char* scsierror(int asc, int ascq)
int scsiverbose;

DESCRIPTION
These routines provide an interface to a SCSI or ATAPI device via sd(3).
Openscsi attempts to open the file devdir/raw and use it to send raw SCSI commands. On success, it reads the device’s inquiry string and stores it in inquire in the returned Scsi structure. Closescsi closes the connection and frees the Scsi structure.
Scsiready sends the “unit ready” command up to three times, returning zero if the unit responds that it is ready, or −1 on error.
Scsierror returns a textual description of the SCSI status denoted by the ASC and ASCQ sense codes. The description is found by consulting /sys/lib/scsicodes. The returned string will be overwritten by the next call to scsierror.
Scsi and scsicmd execute a single SCSI command on the named device. There should be ncmd bytes of command data in cmd; if dir is Sread, a successful operation will store up to ndata bytes into data, returning the number of bytes stored. If dir is Swrite, the ndata bytes beginning at data are transmitted as the data argument to the command, and the number of bytes written is returned. If dir is Snone, data and ndata are ignored. On error, scsi and scsicmd return −1. Scsicmd simply issues the command and returns the result; scsi works a bit harder and is the more commonly used routine. Scsi attempts to send the command; if it is successful, scsi returns what scsicmd returned. Otherwise, scsi sends a request sense command to obtain the reason for the failure, sends a unit ready command in an attempt to bring the unit out of any inconsistent states, and tries again. If the second try fails, scsi sends the request sense and unit ready commands again and then uses scsierror to set errstr with a reason for failure.

The nchange and changetime fields in the Scsi structure record the number of times a media change has been detected, and the time when the current media was inserted into the drive (really the first time a SCSI command was issued after it was inserted). They are maintained by scsi.

If scsiverbose is set, these commands will produce a fair amount of debugging output on file descriptor 2 when SCSI commands fail.

FILES
/sys/lib/scsicodes
   List of textual messages corresponding to SCSI error codes; consulted by scsierror.
SOURCE
/sys/src/libdisk/scsi.c
SEE ALSO
sd(3), scuzz(8)
NAME
md4, md5, shal, sha2_224, sha2_256, sha2_384, sha2_512, aes, hmac_md5, hmac_sha1, hmac_sha2_224, hmac_sha2_256, hmac_sha2_384, hmac_sha2_512, hmac_aes, md5pickle, md5unpickle, shalpickle, shalunpickle – cryptographically secure hashes

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <mp.h>
#include <libsec.h>
define DS DigestState /* only to abbreviate SYNOPSIS */
DS* md4(uchar *data, ulong dlen, uchar *digest, DS *state)
DS* md5(uchar *data, ulong dlen, uchar *digest, DS *state)
char* md5pickle(MD5state *state)
MD5state* md5unpickle(char *p);
DS* shal(uchar *data, ulong dlen, uchar *digest, DS *state)
char* shalpickle(SHA1state *state)
SHA1state* shalunpickle(char *p);
DS* sha2_224(uchar *data, ulong dlen, uchar *digest, DS *state)
DS* sha2_256(uchar *data, ulong dlen, uchar *digest, DS *state)
DS* sha2_384(uchar *data, ulong dlen, uchar *digest, DS *state)
DS* sha2_512(uchar *data, ulong dlen, uchar *digest, DS *state)
DS* aes(uchar *data, ulong dlen, uchar *digest, DS *state)
DS* hmac_x(uchar *p, ulong len, uchar *key, ulong klen, uchar *
digest, DS *state)
char* hmac_md5(uchar *data, ulong dlen, uchar *key, ulong klen, uchar *
digest, DS *state)
char* hmac_sha1(uchar *data, ulong dlen, uchar *key, ulong klen, uchar *
digest, DS *state)
char* hmac_sha2_224(uchar *data, ulong dlen, uchar *key, ulong klen, uchar *
digest, DS *state)
char* hmac_sha2_256(uchar *data, ulong dlen, uchar *key, ulong klen, uchar *
digest, DS *state)
char* hmac_sha2_384(uchar *data, ulong dlen, uchar *key, ulong klen, uchar *
digest, DS *state)
char* hmac_sha2_512(uchar *data, ulong dlen, uchar *key, ulong klen, uchar *
digest, DS *state)
char* hmac_aes(uchar *data, ulong dlen, uchar *key, ulong klen, uchar *
digest, DS *state)

description
We support several secure hash functions. The output of a hash is called a digest. A hash is secure if, given the hashed data and the digest, it is difficult to predict the change to the digest resulting from some change to the data without rehashing the whole data. Therefore, if a secret is part of the hashed data, the digest can be used as an integrity check of the data by anyone possessing the secret.

The routines md4, md5, shal, sha2_224, sha2_256, sha2_384, sha2_512, aes, hmac_md5, hmac_sha1, hmac_sha2_224, hmac_sha2_256, hmac_sha2_384, hmac_sha2_512, and hmac_aes differ only in the length of the resulting digest and in the security of the hash. Sha2_* and hmac_sha2_* are the SHA-2 functions; the number after the final underscore is the number of bits
in the resulting digest. Usage for each is the same. The first call to the routine should have nil as the state parameter. This call returns a state which can be used to chain subsequent calls. The last call should have digest non-nil. Digest must point to a buffer of at least the size of the digest produced. This last call will free the state and copy the result into digest.

The constants MD4dlen, MD5dlen, SHA1dlen, SHA2_224dlen, SHA2_256dlen, SHA2_384dlen, SHA2_512dlen, and AESdlen define the lengths of the digests.

Hmac_md5, hmac_sha1, hmac_sha2_224, hmac_sha2_256, hmac_sha2_384, hmac_sha2_512, and hmac_aes are used slightly differently. These hash algorithms are keyed and require a key to be specified on every call. The digest lengths for these hashes are the obvious ones from the above list of length constants. These routines all call hmac_x internally, but hmac_x is not intended for general use.

The functions md5pickle and sha1pickle marshal the state of a digest for transmission. Md5unpickle and sha1unpickle unmarshal a pickled digest. All four routines return a pointer to a newly malloc(2)’d object.

**EXAMPLES**

To hash a single buffer using md5:

```c
uchar digest[MD5dlen];
md5(data, len, digest, nil);
```

To chain a number of buffers together, bounded on each end by some secret:

```c
char buf[256];
uchar digest[MD5dlen];
DigestState *s;

s = md5("my password", 11, nil, nil);
while((n = read(fd, buf, 256)) > 0)
    md5(buf, n, nil, s);
md5("drowssap ym", 11, digest, s);
```

**SOURCE**

/sys/src/libsec

**SEE ALSO**

aes(2), blowfish(2), des(2), elgamal(2), rc4(2), rsa(2)
/lib/rfc/rfc2104  HMAC specification
NAME
seek – change file offset

SYNOPSIS
#include <u.h>
#include <libc.h>
vlong seek(int fd, vlong n, int type)

DESCRIPTION
Seek sets the offset for the file associated with fd as follows:
   If type is 0, the offset is set to n bytes.
   If type is 1, the pointer is set to its current location plus n.
   If type is 2, the pointer is set to the size of the file plus n.
   The new file offset value is returned.
   Seeking in a directory is not allowed. Seeking in a pipe is a no-op.

SOURCE
/sys/src/libc/9syscall

SEE ALSO
intro(2), open(2)

DIAGNOSTICS
Sets errstr.
NAME
segattach, segdetach, segfree - map/unmap a segment in virtual memory

SYNOPSIS
#include <u.h>
#include <libc.h>
void*segattach(int attr, char *class, void *va, ulong len)
int segdetach(void *addr)
int segfree(void *va, ulong len)

DESCRIPTION
Segattach creates a new memory segment, adds it to the calling process’s address space, and returns its lowest address. Segments belong to system-dependent classes. Segment classes memory (plain memory) and shared (shared memory) are available on all systems.

Shared segments are inherited by the children of the attaching process and remain untouched across a fork(2). An exec(2) will release a shared segment if it overlaps the segments in the file being exec’ed; otherwise the segment will be inherited.

Some machines provide a segment class lock. Lock segments allow access to special lock hardware provided by some multiprocessors, in particular the SGI Power Series machines.

Systems may also provide interfaces to special hardware devices like frame buffers through the segattach interface. Device memory mapped by this method is typically uncached by default.

If the specified class is unknown, segattach draws an error.

Attr specifies the new segment’s attributes. The only attributes implemented on all classes of segment is SG_RONLY, which allows only read access on the segment, and SG_CEXEC, which causes the segment to be detached when the process does an exec(2). Specific devices may implement attributes to control caching and allocation, but these will vary between devices.

Va and len specify the position of the segment in the process’s address space. Va is rounded down to the nearest page boundary and va+len is rounded up. The system does not permit segments to overlap. If va is zero, the system will choose a suitable address.

Segdetach removes a segment from a process’s address space. Memory used by the segment is freed. Addr may be any address within the bounds of the segment.

The system will not permit the initial stack segment to be detached from the address space.

Segfree tells the system that it may free any physical memory within the span [va, va+len], but leaves that portion of the process’s address space valid. The system will not free any memory outside that span, and may not free all or any of the specified memory. If free’d memory is later referenced, it will be initialized as appropriate for the segment type. For example data and text segments will be read from the executable file, and bss segments will be filled with zero bytes.

The MIPS R2000 and R3000 have no hardware instructions to implement locks. The following method can be used to build them from software. First, try to segattach a segment of class lock. If this succeeds, the machine is an SGI Power Series and the memory contains hardware locks. Each 4096-byte page has 64 long words at its beginning; each word implements a test--and--set semaphore when read; the low bit of the word is zero on success, one on failure. If the segattach fails, there is no hardware support but the operating system helps: Any COP3 instruction will be trapped by the kernel and interpreted as a test--and--set. In the trap, R1 points to a long; on return, R1 is greater or equal zero on success, negative on failure. The following assembly language implements such a test--and--set.

/*
*    MIPS test and set
*/

TEXT tas(SB), $0
MOVW R1, sema+0(FP) /* save arg on stack */
btas:
MOVW sema+0(FP), R1
MOVB R0, 1(R1)
NOR R0, R0, R0 /* NOP */
WORD $(023<<26) /* MFC3 R0, R0 */
BLTZ R1, btas
RET

SOURCE
/sys/src/libc/9syscall

SEE ALSO
lock(2), segbrk(2), segflush(2)
/proc/*/segment

DIAGNOSTICS
These functions set errstr. Segattach returns (void*)-1 on error.

BUGS
There is a small fixed limit on the number of segments that may be attached, as well as a maximum segment size.
NAME
segbrk - change memory allocation

SYNOPSIS
#include <u.h>
#include <libc.h>
void* segbrk(void *saddr, void *addr)

DESCRIPTION
Segbrk sets the system's idea of the lowest unused location of a segment to addr rounded up to
the next multiple of a page size, typically 4096 bytes. The segment is identified by saddr which
may be any valid address within the segment.

A call to segbrk with a zero addr argument returns the address of the top of bss.

The system will prevent segments from overlapping and will not allow the length of the text, data,
or stack segment to be altered.

SOURCE
/sys/src/libc/9syscall

SEE ALSO
brk(2), segattach(2), segflush(2)
/proc/*/segment

DIAGNOSTICS
Sets errstr. Segbrk returns (void*)−1 on error.

BUGS
Segbrk is not fully defined or implemented. In particular, it cannot always return the top of bss
when called with a zero addr argument. The segbrk system call may go away or be re-
implemented to give more general segment control, subsuming the functions of brk(2),
segflush(2) and segfree in segattach(2).
NAME
segflush - flush instruction and data caches

SYNOPSIS
#include <u.h>
#include <libc.h>
int segflush(void *va, ulong len)

DESCRIPTION
Segflush invalidates any instruction cache and writes back any data cache associated with pages contained in a segment. All subsequent new pages in the segment will also be flushed when first referenced.

Va is an address within the segment to be flushed; it is rounded down to the nearest page boundary. Len specifies the length in bytes of the memory to flush; va+len is rounded up to the nearest page boundary. Segflush works correctly when the memory straddles multiple segments.

Correct use of segflush depends on an understanding of the cache architecture of the specific machine.

SOURCE
/sys/src/libc/9syscall

SEE ALSO
segattach(2), segbrk(2)
/proc/*/segment

DIAGNOSTICS
Sets errstr.
NAME
  semacquire, semrelease - user level semaphores

SYNOPSIS
  #include <u.h>
  #include <libc.h>
  int semacquire(long *addr, int block);
  long semrelease(long *addr, long count);

DESCRIPTION
  Semacquire and semrelease facilitate scheduling between processes sharing
  memory. Processes arrange to share memory by using rfork with the RFMEM flag
  (see fork(2)), segattach(2), or thread(2).

  The semaphore's value is the integer pointed at by addr. Semacquire
  atomically waits until the semaphore has a positive value and then
  decrements that value. It returns 1 if the semaphore was
  acquired and -1 on error (e.g., if it was interrupted). If block
  is zero and the semaphore is not immediately available, semacquire
  returns 0 instead of waiting. Semrelease adds count to the
  semaphore's value and returns the new value.

  Semacquire and semrelease can be thought of as efficient, correct
  replacements for:

  int
     semacquire(long *addr, int block)
     {
       while(*addr == 0){
         if(!block)
           return 0;
         if(interrupted)
           return -1;
       }
       --*addr;
       return 1;
     }

  int
     semrelease(long *addr, int count)
     {
       return *addr += count;
     }

  Like rendezvous(2), semacquire and semrelease are not typically used
directly. Instead, they are intended to be used to coordinate
scheduling in higher-level abstractions such as locks, rendezvous
points, and channels (see lock(2) and thread(2)). Also like
rendezvous, semacquire and semrelease cannot be used to coordinate
between threads in a single process. Use locks, rendezvous
points, or channels instead.

SOURCE
  /sys/src/9/port/sysproc.c

SEE ALSO
  fork(2), lock(2), rendezvous(2), segattach(2), thread(2)

DIAGNOSTICS
  These functions set errstr.
NAME
setjmp, longjmp, notejmp - non-local goto

SYNOPSIS
#include <u.h>
#include <libc.h>
int setjmp(jmp_buf env)
void longjmp(jmp_buf env, int val)
void notejmp(void *uregs, jmp_buf env, int val)

DESCRIPTION
These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in env for later use by longjmp. It returns value 0.

Longjmp restores the environment saved by the last call of setjmp. It then causes execution to continue as if the call of setjmp had just returned with value val. The invoker of setjmp must not itself have returned in the interim. All accessible data have values as of the time longjmp was called.

Notejmp is the same as longjmp except that it is to be called from within a note handler (see notify(2)). The uregs argument should be the first argument passed to the note handler.

Setjmp and longjmp can also be used to switch stacks. Several macros are defined in /$objtype/include/u.h that can be used to build jmp_bufs by hand. The following code establishes a jmp_buf that may be called by longjmp to begin execution in a function f with 1024 bytes of stack:

#include <u.h>
#include <libc.h>

jmp_buf label;
#define NSTACK 1024
char stack[NSTACK];

void
setLabel(void)
{
    label[JMPBUFPC] = ((ulong)f+JMPBUFDPC);
    /* -2 leaves room for old pc and new pc in frame */
    label[JMPBUFSP] =
        (ulong)(&stack[NSTACK-2*sizeof(ulong*)]);
}

SOURCE
/sys/src/libc/$objtype/setjmp.s
/sys/src/libc/$objtype/notejmp.c

SEE ALSO
notify(2)

BUGS
Notejmp cannot recover from an address trap or bus error (page fault) on the 680x0 architectures.
NAME
    sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

SYNOPSIS
    #include <u.h>
    #include <libc.h>
    double sin(double x)
    double cos(double x)
    double tan(double x)
    double asin(double x)
    double acos(double x)
    double atan(double x)
    double atan2(double y, double x)

DESCRIPTION
    Sin, cos and tan return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.
    Asin returns the arc sine in the range $-\pi/2$ to $\pi/2$.
    Acos returns the arc cosine in the range 0 to $\pi$.
    Atan returns the arc tangent in the range $-\pi/2$ to $\pi/2$.
    Atan2 returns the arc tangent of $y/x$ in the range $-\pi$ to $\pi$.

SOURCE
    /sys/src/libc/port

SEE ALSO
    intro(2)

BUGS
    The value of tan for arguments greater than about $2^{31}$ is garbage.
NAME
    sinh, cosh, tanh – hyperbolic functions

SYNOPSIS
    #include <u.h>
    #include <libc.h>
    double sinh(double x)
    double cosh(double x)
    double tanh(double x)

DESCRIPTION
    These functions compute the designated hyperbolic functions for real arguments.

SOURCE
    /sys/src/libc/port

SEE ALSO
    intro(2)
NAME
    sleep, alarm - delay, ask for delayed note
SYNOPSIS
    #include <u.h>
    #include <libc.h>
    int sleep(long millisecs)
    long alarm(unsigned long millisecs)
DESCRIPTION
    Sleep suspends the current process for the number of milliseconds specified by the argument. The actual suspension time may be a little more or less than the requested time. If millisecs is 0, the process gives up the CPU if another process is waiting to run, returning immediately if not. Sleep returns -1 if interrupted, 0 otherwise.
    Alarm causes an alarm note (see notify(2)) to be sent to the invoking process after the number of milliseconds given by the argument. Successive calls to alarm reset the alarm clock. A zero argument clears the alarm. The return value is the amount of time previously remaining in the alarm clock.
SOURCE
    /sys/src/libc/9syscall
SEE ALSO
    intro(2)
DIAGNOSTICS
    These functions set errstr.
NAME
stat, fstat, wstat, fwstat, dirstat, dirfstat, dirwstat, dirfwstat, nulldir – get and put file status

SYNOPSIS
#include <u.h>
#include <libc.h>
int stat(char *name, uchar *edir, int nedir)
int fstat(int fd, uchar *edir, int nedir)
int wstat(char *name, uchar *edir, int nedir)
int fwstat(int fd, uchar *edir, int nedir)
Dir* dirstat(char *name)
Dir* dirfstat(int fd)
int dirwstat(char *name, Dir *dir)
int dirfwstat(int fd, Dir *dir)
void nulldir(Dir *d)

DESCRIPTION
Given a file’s name, or an open file descriptor fd, these routines retrieve or modify file status information. Stat, fstat, wstat, and fwstat are the system calls; they deal with machine-independent directory entries. Their format is defined by stat(5). Stat and fstat retrieve information about name or fd into edir, a buffer of length nedir, defined in <libc.h>. Wstat and fwstat write information back, thus changing file attributes according to the contents of edir. The data returned from the kernel includes its leading 16-bit length field as described in intro(5). For symmetry, this field must also be present when passing data to the kernel in a call to wstat and fwstat, but its value is ignored.

Dirstat, dirfstat, dirwstat, and dirfwstat are similar to their counterparts, except that they operate on Dir structures:

typedef
struct Dir {
    /* system-modified data */
    uint type;  /* server type */
    uint dev;   /* server subtype */
    /* file data */
    Qid qid;    /* unique id from server */
    ulong mode; /* permissions */
    ulong atime; /* last read time */
    ulong mtime; /* last write time */
    vlong length; /* file length: see <u.h> */
    char *name; /* last element of path */
    char *uid;  /* owner name */
    char *gid;  /* group name */
    char *muid; /* last modifier name */
}

The returned structure is allocated by malloc(2); freeing it also frees the associated strings.

This structure and the Qid structure are defined in <libc.h>. If the file resides on permanent storage and is not a directory, the length returned by stat is the number of bytes in the file. For directories, the length returned is zero. For files that are streams (e.g., pipes and network connections), the length is the number of bytes that can be read without blocking.

Each file is the responsibility of some server: it could be a file server, a kernel device, or a user process. Type identifies the server type, and dev says which of a group of servers of the same type is the one responsible for this file. Qid is a structure containing path and vers fields: path is guaranteed to be unique among all path names currently on the file server, and vers changes each time the file is modified. The path is a long long (64 bits, vlong) and the vers is an unsigned long (32 bits, ulong). Thus, if two files have the same type, dev,
and qid they are the same file.

The bits in mode are defined by

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000000</td>
<td>directory</td>
</tr>
<tr>
<td>0x40000000</td>
<td>append only</td>
</tr>
<tr>
<td>0x20000000</td>
<td>exclusive use (locked)</td>
</tr>
</tbody>
</table>

0400  read permission by owner
0200  write permission by owner
0100  execute permission (search on directory) by owner
0070  read, write, execute (search) by group
0007  read, write, execute (search) by others

There are constants defined in `<libc.h>` for these bits: DMDIR, DMAPPEND, and DMEXCL for the first three; and DMREAD, DMWRITE, and DMEXEC for the read, write, and execute bits for others.

The two time fields are measured in seconds since the epoch (Jan 1 00:00 1970 GMT). Mtime is the time of the last change of content. Similarly, atime is set whenever the contents are accessed; also, it is set whenever mtime is set.

Uid and gid are the names of the owner and group of the file; muid is the name of the user that last modified the file (setting mtime). Groups are also users, but each server is free to associate a list of users with any user name g, and that list is the set of users in the group g. When an initial attachment is made to a server, the user string in the process group is communicated to the server. Thus, the server knows, for any given file access, whether the accessing process is the owner of, or in the group of, the file. This selects which sets of three bits in mode is used to check permissions.

Only some of the fields may be changed with the `wstat` calls. The name can be changed by anyone with write permission in the parent directory. The mode and mtime can be changed by the owner or the group leader of the file's current group. The gid can be changed: by the owner if also a member of the new group; or by the group leader of the file's current group if also leader of the new group (see `intro(5)` for more information about permissions and `users(6)` for users and groups). The length can be changed by anyone with write permission, provided the operation is implemented by the server. (See `intro(5)` for permission information, and `users(6)` for user and group information).

Special values in the fields of the Dir passed to `wstat` indicate that the field is not intended to be changed by the call. The values are the maximum unsigned integer of appropriate size for integral values (usually ~0, but beware of conversions and size mismatches when comparing values) and the empty or nil string for string values. The routine `nulldir` initializes all the elements of d to these "don't care" values. Thus one may change the mode, for example, by using `nulldir` to initialize a Dir, then setting the mode, and then doing `wstat`; it is not necessary to use `stat` to retrieve the initial values first.

**SOURCE**

```
/sys/src/libc/9syscall  for the non-dir routines
/sys/src/libc/9sys     for the routines prefixed dir
```

**SEE ALSO**

`intro(2)`, `fcall(2)`, `dirread(2)`, `stat(5)`

**DIAGNOSTICS**

The `dir` functions return a pointer to the data for a successful call, or nil on error. The others return the number of bytes copied on success, or -1 on error. All set `errno`.

If the buffer for `stat` or `fstat` is too short for the returned data, the return value will be BIT16SZ (see `fcall(2)`) and the two bytes returned will contain the initial count field of the returned data; retrying with `nedir` equal to that value plus BIT16SZ (for the count itself) should succeed.
NAME
strcat, strncat, strcmp, strncmp, cistrcmp, cistrcncmp, strcpy, strncpy, strecpy, strlcn, strchr, strrchr, strpbrk, strspn, strcspsn, strtok, strdup, strstr, cistrstr – string operations

SYNOPSIS
#include <u.h>
#include <libc.h>
char* strcat(char *s1, char *s2)
char* strncat(char *s1, char *s2, long n)
int strcmp(char *s1, char *s2)
int strnncmp(char *s1, char *s2, long n)
int cistrcmp(char *s1, char *s2)
int cistrcncmp(char *s1, char *s2, long n)
char* strcpy(char *s1, char *s2)
char* strecpy(char *s1, char *es1, char *s2)
char* strncpy(char *s1, char *s2, long n)
long strlen(char *s)
char* strchr(char *s, char c)
char* strrrchr(char *s, char c)
char* strpbrk(char *s1, char *s2)
long strspn(char *s1, char *s2)
long strcspsn(char *s1, char *s2)
char* strtok(char *s1, char *s2)
char* strdup(char *s)
char* strstr(char *s1, char *s2)
char* cistrstr(char *s1, char *s2)

DESCRIPTION
The arguments s1, s2 and s point to null-terminated strings. The functions strcat, strncat, strcprüp, strecpy, and strncpy all alter s1. Strcat and strecpy do not check for overflow of the array pointed to by s1.

Strcat appends a copy of string s2 to the end of string s1. Strncat appends at most n bytes. Each returns a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer less than, equal to, or greater than 0, according as s1 is lexicographically less than, equal to, or greater than s2. Strncmp makes the same comparison but examines at most n bytes. Cistrcmp and cistrncmp ignore ASCII case distinctions when comparing strings. The comparisons are made with unsigned bytes.

Strecpy copies string s2 to s1, stopping after the null byte has been copied. Strncpy copies exactly n bytes, truncating s2 or adding null bytes to s1 if necessary. The result will not be null-terminated if the length of s2 is n or more. Each function returns s1.

Strecpy copies bytes until a null byte has been copied, but writes no bytes beyond es1. If any bytes are copied, s1 is terminated by a null byte, and a pointer to that byte is returned. Otherwise, the original s1 is returned.

Strlen returns the number of bytes in s, not including the terminating null byte.

Strchr (strrrchr) returns a pointer to the first (last) occurrence of byte c in string s, or 0 if c does not occur in the string. The null byte terminating a string is considered to be part of the string.

Strpbrk returns a pointer to the first occurrence in string s1 of any byte from string s2, 0 if no byte from s2 exists in s1.
Strspn (strcspn) returns the length of the initial segment of string s1 which consists entirely of bytes from (not from) string s2.

Strtok considers the string s1 to consist of a sequence of zero or more text tokens separated by spans of one or more bytes from the separator string s2. The first call, with pointer s1 specified, returns a pointer to the first byte of the first token, and will have written a null byte into s1 immediately following the returned token. The function keeps track of its position in the string between separate calls; subsequent calls, signified by s1 being 0, will work through the string s1 immediately following that token. The separator string s2 may be different from call to call. When no token remains in s1, 0 is returned.

Strdup returns a pointer to a distinct copy of the null-terminated string s in space obtained from malloc(2) or 0 if no space can be obtained.

Strstr returns a pointer to the first occurrence of s2 as a substring of s1, or 0 if there is none. If s2 is the null string, strstr returns s1. Cistrstr operates analogously, but ignores ASCII case differences when comparing strings.

SOURCE
All these routines have portable C implementations in /sys/src/libc/port. Many also have machine-dependent assembly language implementations in /sys/src/libc/$objtype.

SEE ALSO
memory(2), rune(2), runestrcat(2), string(2)

BUGS
These routines know nothing about UTF. Use the routines in rune(2) as appropriate. Note, however, that the definition of UTF guarantees that strcmp compares UTF strings correctly.

The outcome of overlapping moves varies among implementations.
NAME
s_alloc, s_append, s_array, s_copy, s_error, s_free, s_incref, s_memappend, s_nappend, s_new,
s_newalloc, s_parse, s_reset, s_restart, s_terminate, s_tolower, s_putc, s_unique, s_grow, s_read,
s_read_line, s_getline - extensible strings

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <String.h>

String* s_new(void)
void s_free(String *s)
String* s_newalloc(int n)
String* s_array(char *p, int n)
String* s_grow(String *s, int n)
void s_putc(String *s, int c)
void s_terminate(String *s)
String* s_reset(String *s)
String* s_restart(String *s)
String* s_append(String *s, char *p)
String* s_nappend(String *s, char *p, int n)
String* s_memappend(String *s, char *p, int n)
String* s_copy(char *p)
String* s_parse(String *s1, String *s2)
void s_tolower(String *s)
String* s_incref(String *s)
String* s_unique(String *s)

#include <bio.h>

int s_read(Biobuf *b, String *s, int n)
char* s_read_line(Biobuf *b, String *s)
char* s_getline(Biobuf *b, String *s)

DESCRIPTION
These routines manipulate extensible strings. The basic type is String, which points to an array
of characters. The string maintains pointers to the beginning and end of the allocated array. In
addition a finger pointer keeps track of where parsing will start (for s_parse) or new characters will
be added (for s_putc, s_append, and s_nappend). The structure, and a few useful macros are:

typedef struct String {
   Lock;
   char *base;  /* base of String */
   char *end;   /* end of allocated space+1 */
   char *ptr;   /* ptr into String */
... } String;

#define s_to_c(s) ((s)->base)
#define s_len(s) ((s)->ptr-(s)->base)
#define s_clone(s) s_copy((s)->base)

S_to_c is used when code needs a reference to the character array. Using s->base directly is
frowned upon since it exposes too much of the implementation.

allocation and freeing
A string must be allocated before it can be used. One normally does this using s_new, giving the
string an initial allocation of 128 bytes. If you know that the string will need to grow much longer,
you can use s_newalloc instead, specifying the number of bytes in the initial allocation.

s_free causes both the string and its character array to be freed.
_S_.grow grows a string’s allocation by a fixed amount. It is useful if you are reading directly into a string’s character array but should be avoided if possible.

_S_.array is used to create a constant array, that is, one whose contents won’t change. It points directly to the character array given as an argument. Tread lightly when using this call.

**Filling the string**

After its initial allocation, the string points to the beginning of an allocated array of characters starting with NUL.

_S_.putc writes a character into the string at the pointer and advances the pointer to point after it.

_S_.terminate writes a NUL at the pointer but doesn’t advance it.

_S_.restart resets the pointer to the beginning of the string but doesn’t change the contents.

_S_.reset is equivalent to _S_.restart followed by _S_.terminate.

_S_.append and _S_.nappend copy characters into the string at the pointer and advance the pointer. They also write a NUL at the pointer without advancing the pointer beyond it. Both routines stop copying on encountering a NUL. _S_.memappend is like _S_.nappend but doesn’t stop at a NUL.

If you know the initial character array to be copied into a string, you can allocate a string and copy in the bytes using _S_.copy. This is the equivalent of a _S_.new followed by an _S_.append.

_S_.parse copies the next white space terminated token from _s1_ to the end of _s2_. White space is defined as space, tab, and newline. Both single and double quoted strings are treated as a single token. The bounding quotes are not copied. There is no escape mechanism.

_S_.tolower converts all ASCII characters in the string to lower case.

**Multithreading**

_S_.incRef is used by multithreaded programs to avoid having the string memory released until the last user of the string performs an _S_.free. _S_.unique returns a unique copy of the string: if the reference count it 1 it returns the string, otherwise it returns an _S_.clone of the string.

**Bio interaction**

_S_.read reads the requested number of characters through a _Biobuf_ into a string. The string is grown as necessary. An eof or error terminates the read. The number of bytes read is returned. The string is null terminated.

_S_.read_line reads up to and including the next newline and returns a pointer to the beginning of the bytes read. An eof or error terminates the read. The string is null terminated.

_S_.getline reads up to the next newline and returns a pointer to the beginning of the bytes read. Leading spaces and tabs and the trailing newline are all discarded. _S_.getline will recursively read through files included with #include and discard all other lines beginning with #.

**SOURCE**

/sys/src/libString

**SEE ALSO**

_bio(2)_
NAME
stringsize, stringwidth, stringnwidth, runestringsize, runestringwidth, runestringnwidth – graphical
size of strings

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>

Point stringsize(Font *f, char *s)
int stringwidth(Font *f, char *s)
int stringnwidth(Font *f, char *s, int n)
Point runestringsize(Font *f, Rune *s)
int runestringwidth(Font *f, Rune *s)
int runestringnwidth(Font *f, Rune *s, int n)

DESCRIPTION
These routines compute the geometrical extent of character strings when drawn on the display. The most
straightforward, stringsize, returns a Point representing the vector from upper left to lower right of the
NUL-terminated string s drawn in font f. Stringwidth returns just the
x component. Stringnwidth returns the width of the first n characters of s.
The routines beginning with rune are analogous, but accept an array of runes rather than UTF-
encoded bytes.

FILES
/lib/font/bit directory of fonts

SOURCE
/sys/src/libdraw

SEE ALSO
addpt(2), cachechars(2), subfont(2), draw(2), draw(3), image(6), font(6)

DIAGNOSTICS
Because strings are loaded dynamically, these routines may generate I/O to the server and produce
calls to the graphics error function.
NAME
 allocsubfont, freesubfont, installsubfont, lookupsubfont, uninstallsubfont, subfontname, readsubfont, readsubfonti, writesubfont, stringsubfont, strsubfontwidth, mkfont - subfont manipulation

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
Subfont* allocsubfont(char *name, int n, int height, int ascent, Fontchar *info, Image *i)
void freesubfont(Subfont *f)
void installsubfont(char *name, Subfont *f)
Subfont* lookupsubfont(Subfont *f)
Subfont* readsubfont(Display *d, char *name, int fd, int dolock)
Subfont* readsubfonti(Display *d, char *name, int fd, Image *im, int dolock)
int writesubfont(int fd, Subfont *f)
Point stringsubfont(Image *dst, Point p, Image *src, Subfont *f, char *str)
Point strsubfontwidth(Subfont *f, char *s)
Font* mkfont(Subfont *f, Rune min)

DESCRIPTION
Subfonts are the components of fonts that hold the character images. A font comprises an array of subfonts; see cachechars(2). A new Subfont is allocated and initialized with allocsubfont. See cachechars(2) for the meaning of n, height, ascent, and info, and the arrangement of characters in image i. The name is used to identify the subfont in the subfont cache; see the descriptions lookupsubfont and installsubfont (q.v.). The appropriate fields of the returned Subfont structure are set to the passed arguments, and the image is registered as a subfont with the graphics device draw(3). Allocsubfont returns 0 on failure.

Freesubfont frees a subfont and all its associated structure including the associated image. Since freesubfont calls free on f->info, if f->info was not allocated by malloc(2) it should be zeroed before calling subfree.

A number of subfonts are kept in external files. The convention for naming subfont files is:

    /lib/font/bit/name/class.size.depth

where size is approximately the height in pixels of the lower case letters (without ascenders or descenders). If there is only one version of the subfont, the .depth extension is elided. Class describes the range of runes encoded in the subfont: ascii, latin1, greek, etc.

Subfonts are cached within the program, so a subfont shared between fonts will be loaded only once. Installsubfont stores subfont f under the given name, typically the file name from which it was read. Uninstallsubfont removes the subfont from the cache. Finally, lookupsubfont searches for a subfont with the given name in the cache and returns it, or nil if no such subfont exists.

Subfontname is used to locate subfonts given their names within the fonts. The default version constructs a name given the cfname, its name within the font, fname, the name of the font, and the maximum depth suitable for this subfont. This interface allows a partially specified name within a font to be resolved at run-time to the name of a file holding a suitable subfont. Although it is principally a routine internal to the library, subfontname may be substituted by the application to provide a less file-oriented subfont naming scheme.

The format of a subfont file is described in font(6). Briefly, it contains a image with all the characters in it, followed by a subfont header, followed by character information. Readsubfont reads a subfont from the file descriptor fd. The name is used to identify the font in the cache. The dolock
argument specifies whether the routine should synchronize use of the Display with other processes; for single-threaded applications it may always be zero. Readsubfonti does the same for a subfont whose associated image is already in memory; it is passed as the argument im. In other words, readsubfonti reads only the header and character information from the file descriptor.

Writesubfont writes on fd the part of a subfont file that comes after the image. It should be preceded by a call to writeimage (see allocimage(2)).

Strings subfont is analogous to string (see draw(2)) for subfonts. Rather than use the underlying font caching primitives, it calls draw for each character. It is intended for stand-alone environments such as operating system kernels. Strsubfontwidth returns the width of the string s in as it would appear if drawn with stringsubfont in SubFont f.

Mkfont takes as argument a SubFont s and returns a pointer to a Font that maps the character images in s into the Runes min to min+s->n−1.

FILES
/lib/font/bit bitmap font file tree

SOURCE
/sys/src/libdraw

SEE ALSO
  graphics(2), allocimage(2), draw(2), cachechars(2), image(6), font(6)

DIAGNOSTICS
  All of the functions use the graphics error function (see graphics(2)).
NAME
syminit, getsym, symbase, pc2sp, pc2line, textseg, line2addr, lookup, findlocal, getauto, findsym, localsym, globalsym, textsym, file2pc, fileelem, filesym, fileline, fnbound – symbol table access functions

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <bio.h>
#include <mach.h>
int syminit(int fd, Fhdr *fp)
Sym *gtsym(int index)
Sym *symbase(long *nsyms)
int fileelem(Sym **fp, uchar *encname, char *buf, int n)
int filesym(int index, char *buf, int n)
long pc2sp(ulong pc)
long pc2line(ulong pc)
void textseg(ulong base, Fhdr *fp)
long line2addr(ulong line, ulong basepc)
int lookup(char *fn, char *var, Symbol *s)
int findlocal(Symbol *s1, char *name, Symbol *s2)
int getauto(Symbol *s1, char *name, Symbol *s2)
int findsym(long addr, int class, Symbol *s)
int localsym(Symbol *s, int index)
int globalsym(Symbol *s, int index)
int textsym(Symbol *s, int index)
long file2pc(char *file, ulong line)
int fileline(char *str, int n, ulong addr)
int fnbound(char *str, int n, ulong *bounds)

DESCRIPTION
These functions provide machine-independent access to the symbol table of an executable file or executing process. The latter is accessible by opening the device /proc/pid/text as described in proc(3). Mach(2) and object(2) describe additional library functions for processing executable and object files.

Syminit, getsym, symbase, fileelem, pc2sp, pc2line, and line2addr process the symbol table contained in an executable file or the text image of an executing program. The symbol table is stored internally as an array of Sym data structures as defined in a.out(6).

Syminit uses the data in the Fhdr structure filled by crackhdr (see mach(2)) to read the raw symbol tables from the open file descriptor fd. It returns the count of the number of symbols or -1 if an error occurs.

Getsym returns the address of the ith Sym structure or zero if index is out of range.

Symbase returns the address of the first Sym structure in the symbol table. The number of entries in the symbol table is returned in nsyms.

Fileelem converts a file name, encoded as described in a.out(6), to a character string. Fp is the base of an array of pointers to file path components ordered by path index. Encname is the address of an array of encoded file path components in the form of a z symbol table entry. Buf and n specify the address of a receiving character buffer and its length. Fileelem returns the length of the null-terminated string that is at most n-1 bytes long.
Filesym is a higher-level interface to fileelem. It fills buf with the name of the ith file and returns the length of the null-terminated string that is at most n-1 bytes long. File names are retrieved in no particular order, although the order of retrieval does not vary from one pass to the next. A zero is returned when index is too large or too small or an error occurs during file name conversion.

Pc2sp returns an offset associated with a given value of the program counter. Adding this offset to the current value of the stack pointer gives the address of the current stack frame. This approach only applies to the 68020 architecture; other architectures use a fixed stack frame offset by a constant contained in a dummy local variable (called .frame) in the symbol table.

Pc2line returns the line number of the statement associated with the instruction address pc. The line number is the absolute line number in the source file as seen by the compiler after preprocessing; the original line number in the source file may be derived from this value using the history stacks contained in the symbol table.

Pc2sp and pc2line must know the start and end addresses of the text segment for proper operation. These values are calculated from the file header by function syminit. If the text segment address is changed, the application program must invoke textseg to recalculate the boundaries of the segment. Base is the new base address of the text segment and fp points to the Fhdr data structure filled by crackhdr.

Line2addr converts a line number to an instruction address. The first argument is the absolute line number in a file. Since a line number does not uniquely identify an instruction location (e.g., every source file has line 1), a second argument specifies a text address from which the search begins. Usually this is the address of the first function in the file of interest.

Pc2sp, pc2line, and line2addr return -1 in the case of an error.

Lookup, findlocal, getauto, findsym, localsym, globalsym, textsym, file2pc, and fileline operate on data structures riding above the raw symbol table. These data structures occupy memory and impose a startup penalty but speed retrievals and provide higher-level access to the basic symbol table data. Syminit must be called prior to using these functions. The Symbol data structure:

```c
typedef struct {
    void *handle; /* private */
    struct {
        char *name;
        long value;
        char type;
        char class;
    }
} Symbol;
```

describes a symbol table entry. The value field contains the offset of the symbol within its address space: global variables relative to the beginning of the data segment, text beyond the start of the text segment, and automatic variables and parameters relative to the stack frame. The type field contains the type of the symbol as defined in a.out(6). The class field assigns the symbol to a general class; CTEXT, CDATA, CAUTO, and CPARAM are the most popular.

Lookup fills a Symbol structure with symbol table information. Global variables and functions are represented by a single name; local variables and parameters are uniquely specified by a function and variable name pair. Arguments fn and var contain the name of a function and variable, respectively. If both are non-zero, the symbol table is searched for a parameter or automatic variable. If only var is zero, the text symbol table is searched for function fn. If only fn is zero, the global variable table is searched for var.

Findlocal fills s2 with the symbol table data of the automatic variable or parameter matching name. S1 is a Symbol data structure describing a function or a local variable; the latter resolves to its owning function.

Getauto searches the local symbols associated with function s1 for an automatic variable or parameter located at stack offset off. Class selects the class of variable: CAUTO or CPARAM. S2 is the address of a Symbol data structure to receive the symbol table information of the desired symbol.
Findsym returns the symbol table entry of type class stored near addr. The selected symbol is a global variable or function with address nearest to and less than or equal to addr. Class specification CDATA searches only the global variable symbol table; class CTEXT limits the search to the text symbol table. Class specification CANY searches the text table first, then the global table.

Localsym returns the ith local variable in the function associated with s. S may reference a function or a local variable; the latter resolves to its owning function. If the ith local symbol exists, s is filled with the data describing it.

Globalsym loads s with the symbol table information of the ith global variable.

Textsym loads s with the symbol table information of the ith text symbol. The text symbols are ordered by increasing address.

File2pc returns a text address associated with line in file file, or -1 on an error.

Fileline converts text address addr to its equivalent line number in a source file. The result, a null terminated character string of the form file:line, is placed in buffer str of n bytes.

Fnbound returns the start and end addresses of the function containing the text address supplied as the first argument. The second argument is an array of two unsigned longs; fnbound places the bounding addresses of the function in the first and second elements of this array. The start address is the address of the first instruction of the function; the end address is the address of the start of the next function in memory, so it is beyond the end of the target function. Fnbound returns 1 if the address is within a text function, or zero if the address selects no function.

Functions file2pc and fileline may produce inaccurate results when applied to optimized code.

Unless otherwise specified, all functions return 1 on success, or 0 on error. When an error occurs, a message describing it is stored in the system error buffer where it is available via errstr.

SOURCE
/sys/src/libmach

SEE ALSO
mach(2), object(2), errstr(2), proc(3), a.out(6)
NAME
alt, chanclose, chancreate, chanfree, chaninit, chanclosing, chanprint, mainstacksize, proccreate,
proCDATA, proexec, proexecl, procrfork, recv, recvP, recvvl, send, sendP, sendul, nbrecv,
nbrecvP, nbrecvul, nbsend, nbsendP, nbsendul, threadcreate, threaddata, threadexits, threadexit-
sall, threadgetgrp, threadgetname, threadint, threadintgrp, threadkill, threadkillgrp, threadmain,
threadnotify, threadid, threadpid, threadsetgrp, threadsetname, threadwaitchan, yield – thread and
proc management

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <thread.h>

typedef enum {
  CHANEND,
  CHANSND,
  CHANRCV,
  CHANNOP,
  CHANNOBLLK,
} ChanOp;

typedef struct Alt Alt;
struct Alt {
  Channel *c; /* channel */
  void *v; /* pointer to value */
  ChanOp op; /* operation */
  char *err; /* did the op fail? */
  /*
   * the next variables are used internally to alt
   * they need not be initialized
   */
  Channel **tag; /* pointer to rendez-vous tag */
  int entryno; /* entry number */
};

void threadmain(int argc, char *argv[])
int mainstacksize
int proccreate(void (*fn)(void*), void *arg, uint stacksize)
int procrfork(void (*fn)(void*), void *arg, uint stacksize,
int rforkflag)
int threadcreate(void (*fn)(void*), void *arg, uint stacksize)
void threadexits(char *status)
void threadexitsall(char *status)
void yield(void)
int threadid(void)
int threadgrp(void)
int threadsetgrp(int group)
int threadpid(int id)
int threadint(int id)
void threadintgrp(int group)
void threadkill(int id)
int threadkillgrp(int group)
void threadsetname(char *name, ...)
char* threadgetname(void)
void** threaddata(void)
void** procdata(void)
int chaninit(Channel *c, int elsize, int nel)
Channel* chancreate(int elsize, int nel)
void chanfree(Channel *c)

int alt(Alt *alts)
int recv(Channel *c, void *v)
void* recvvp(Channel *c)
ulong recvul(Channel *c)
int nbrecv(Channel *c, void *v)
void* nbrecvvp(Channel *c)
ulong nbrecvul(Channel *c)
int send(Channel *c, void *v)
int sendp(Channel *c, void *v)
int sendul(Channel *c, ulong v)
int nbsend(Channel *c, void *v)
int nbsendp(Channel *c, void *v)
int nbsendul(Channel *c, ulong v)
int chanprint(Channel *c, char *fmt, ...)
int chanclose(Channel *c);
int chanclosing(Channel *c);
void procexecl(Channel *cpid, char *file, ...)
void procexec(Channel *cpid, char *file, char *args[])
Channel* threadwaitchan(void)
int threadnotify(int (*f)(void*, char*), int in)

DESCRIPTION

The thread library provides parallel programming support similar to that of the languages Alef and Newsqueak. Threads and procs occupy a shared address space, communicating and synchronizing through channels and shared variables.

A proc is a Plan 9 process that contains one or more cooperatively–scheduled threads. Programs using threads must replace main by threadmain. The thread library provides a main function that sets up a proc with a single thread executing threadmain on a stack of size mainstacksize (default eight kilobytes). To set mainstacksize, declare a global variable initialized to the desired value (e.g., int mainstacksize = 1024).

Creation

Threadcreate creates a new thread in the calling proc, returning a unique integer identifying the thread; the thread executes fn(arg) on a stack of size stacksize. Thread stacks are allocated in shared memory, making it valid to pass pointers to stack variables between threads and procs. Procrfork creates a new proc, and inside that proc creates a single thread as threadcreate would, returning the id of the created thread. Procrfork creates the new proc by calling rfork (see fork(2)) with flags RFPROC|RFMEM|RFNOWAIT|rforkflag. (The thread library depends on all its procs running in the same rendezvous group. Do not include RFREND in rforkflag.) Procreate is identical to procrfork with rforkflag set to zero. Be aware that the calling thread may continue execution before the newly created proc and thread are scheduled. Because of this, arg should not point to data on the stack of a function that could return before the new process is scheduled.

Threadexits terminates the calling thread. If the thread is the last in its proc, threadexits also terminates the proc, using status as the exit status. Threadexitsall terminates all procs in the program, using status as the exit status.

Scheduling

The threads in a proc are coroutines, scheduled non–preemptively in a round–robin fashion. A thread must explicitly relinquish control of the processor before another thread in the same proc is run. Calls that do this are yield, proccreate, procexec, procexec1, threadexit, alt, send, and recv (and the calls related to send and recv—see their descriptions further on), plus these from lock(2): glock, rlock, wlock, rsleep. Procs are scheduled by the operating system. Therefore, threads in different procs can preempt one another in arbitrary ways and should synchronize their actions using qLocks (see lock(2)) or channel communication. System calls such as read(2) block the entire proc; all threads in a proc block until the system call finishes.

As mentioned above, each thread has a unique integer thread id. Thread ids are not reused; they are unique across the life of the program. Threadid returns the id for the current thread. Each
thread also has a thread group id. The initial thread has a group id of zero. Each new thread inherits the group id of the thread that created it. Threadgrp returns the group id for the current thread; threadsetgrp sets it. Threadpid returns the pid of the Plan 9 process containing the thread identified by id, or −1 if no such thread is found.

Threadint interrupts a thread that is blocked in a channel operation or system call. Threadintgrp interrupts all threads with the given group id. Threadkill marks a thread to die when it next relinquishes the processor (via one of the calls listed above). If the thread is blocked in a channel operation or system call, it is also interrupted. Threadkillgrp kills all threads with the given group id. Note that threadkill and threadkillgrp will not terminate a thread that never relinquishes the processor.

Names and per-thread data
Primarily for debugging, threads can have string names associated with them. ThreadGetName returns the current thread’s name; threadsetName sets it. The pointer returned by threadGetName is only valid until the next call to threadGetName.

Threaddata returns a pointer to a per-thread pointer that may be modified by threaded programs for per-thread data. Similarly, procdata returns a pointer to a per-proc pointer.

Executing new programs
Procexcl and proexec are threaded analogues of exec and execl (see exec(2)); on success, they replace the calling thread (which must be the only thread in its proc) and invoke the external program, never returning. On error, they return −1. If cpid is not null, the pid of the invoked program will be sent along cpid once the program has started, or −1 will be sent if an error occurs. Procexcl and proexec will not access their arguments after sending a result along cpid. Thus, programs that malloc the argv passed to proexec can safely free it once they have received the cpid response. Note that the mount point /mnt/temp must exist; proexecl() mount pipes there.

Threadwaitchan returns a channel of pointers to Waitmsg structures (see wait(2)). When an exec’d program exits, a pointer to a Waitmsg is sent to this channel. These Waitmsg structures have been allocated with malloc(2) and should be freed after use.

Channels
A Channel is a buffered or unbuffered queue for fixed-size messages. Procfs and threads send messages into the channel and recv messages from the channel. If the channel is unbuffered, a send operation blocks until the corresponding recv operation occurs and vice versa. Chaninit initializes a Channel for messages of size elsize and with a buffer holding nel messages. If nel is zero, the channel is unbuffered. Chancreate allocates a new channel and initializes it. Chanfree frees a channel that is no longer used. Chanfree can be called by either sender or receiver after the last item has been sent or received. Freeing the channel will be delayed if there is a thread blocked on it until that thread unblocks (but chanfree returns immediately).

Send sends the element pointed at by v to the channel c. If v is null, zeros are sent. Recv receives an element from c and stores it in v. If v is null, the received value is discarded. Send and recv return 1 on success, −1 if interrupted. Nsend and nbrecv behave similarly, but return 0 rather than blocking.

Sendp, nbsendp, sendul, and nbsendul send a pointer or an unsigned long; the channel must have been initialized with the appropriate elsize. Recvp, nbrecv, recvul, and nbrecvul receive a pointer or an unsigned long; they return zero when a zero is received, when interrupted, or (for nbrecv and nbrecvul) when the operation would have blocked. To distinguish between these three cases, use recv or nbrecv.

Alt can be used to recv from or send to one of a number of channels, as directed by an array of Alts structures, each of which describes a potential send or receive operation. In an Alt structure, c is the channel; v the value pointer (which may be null); and op the operation: CHANSND for a send operation, CHANRCV for a recv operation; CHANNOP for no operation (useful when alt is called with a varying set of operations). The array of Alts structures is terminated by an entry with op CHANEND or CHANNOBLK. If at least one Alt structure can proceed, one of them is chosen at random to be executed. Alt returns the index of the chosen structure. If no operations can proceed and the list is terminated with CHANNOBLK, alt returns the index of the terminating CHANNOBLK structure. Otherwise, alt blocks until one of the operations can proceed, eventually returning the index of the structure executes. Alt returns −1 when interrupted. The tag and entryno fields in the Alt structure are used internally by alt and need not be initialized. They
are not used between _alt_ calls.

Chanprint formats its arguments in the manner of _print(2)_ and sends the result to the channel _c_. The string delivered by chanprint is allocated with _malloc(2)_ and should be freed upon receipt.

Chanclose prevents further elements being sent to the channel _c_. After closing a channel, _send_ and _recv_ never block. _Send_ always returns –1. _Recv_ returns –1 if the channel is empty. _Alt_ may choose a CHANSND or CHANRCV that failed because the channel was closed. In this case, the _err_ field of the _Alt_ entry points to an error string stating that the channel was closed and the operation was completed with failure. If all entries have been selected and failed because they were closed, _alt_ returns –1.

Errors, notes and resources
Thread library functions do not return on failure; if errors occur, the entire program is aborted.

Chanclosing returns –1 if no one called _closed_ on the channel, and otherwise the number of elements still in the channel.

Threaded programs should use _threadnotify_ in place of _atnotify_ (see _notify(2))_.

It is safe to use _sysfatal_ (see _perror(2))_ in threaded programs. _Sysfatal_ will print the error string and call _threadexitsall_.

It is safe to use _rfork_ (see _fork(2))_ to manage the namespace, file descriptors, note group, and environment of a single process. That is, it is safe to call _rfork_ with the flags RFNAMEG, RFFDG, RFCFDG, RFNOTEQ, RFENVG, and RFCENVG. (To create new processes, use _proccreate_ and _procrfork_.) As mentioned above, the thread library depends on all procs being in the same rendezvous group; do not change the rendezvous group with _rfork_.

FILES
    /sys/lib/acid/thread useful _acid(1)_ functions for debugging threaded programs.
    /sys/src/libthread/example.c a full example program.
    /mnt/temp a place for _procexec_ to create pipes.

SOURCE
    /sys/src/libthread

SEE ALSO
    _intro(2), iopro(2), lock(2)_
NAME
time, nsec - time in seconds and nanoseconds since epoch

SYNOPSIS
#include <u.h>
#include <libc.h>
long time(long *tp)
vlong nsec(void)

DESCRIPTION
Both time and nsec return the time since the epoch 00:00:00 GMT, Jan. 1, 1970. The return value of the former is in seconds and the latter in nanoseconds. For time, if tp is not zero then *tp is also set to the answer.

These functions work by reading /dev/bintime, opening that file when they are first called.

SOURCE
/sys/src/libc/9sys/time.c
/sys/src/libc/9sys/nsec.c

SEE ALSO
cputime(2), cons(3)

DIAGNOSTICS
Sets errstr.

BUGS
These routines maintain a static file descriptor.
NAME
tmpfile, tmpnam - Stdio temporary files

SYNOPSIS
#include <u.h>
#include <stdio.h>
FILE *tmpfile(void)
char *tmpnam(char *s)

DESCRIPTION
Tmpfile creates a temporary file that will automatically be removed when the file is closed or the
program exits. The return value is a Stdio FILE* opened in update mode (see fopen(2)).
Tmpnam generates a string that is a valid file name and that is not the same as the name of an
existing file. If s is zero, it returns a pointer to a string which may be overwritten by subsequent
calls to tmpnam. If s is non-zero, it should point to an array of at least L_tmpnam (defined in
<stdio.h>) characters, and the answer will be copied there.

FILES
/tmp/tf000000000000 template for tmpfile file names.
/tmp/tn000000000000 template for tmpnam file names.

SOURCE
/sys/src/libstdio

BUGS
The files created by tmpfile are not removed until exits(2) is executed; in particular, they are not
removed on fclose or if the program terminates abnormally.
NAME
usbcmd, classname, closedev, configdev, devctl, finddevs, loaddevstr, matchdevcsp, opendev,
opendevdata, openep, startdevs, uninstall, class, subclass, proto, CSP – USB device driver library

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <thread.h>
#include "../lib/usb.h"
struct Dev {
    Ref;
    char* dir; /* path for the endpoint dir */
    int id; /* usb id for device or ep. number */
    int dfd; /* descriptor for the data file */
    int cfd; /* descriptor for the control file */
    int maxpkt; /* cached from usb description */
    Usbdev* usb; /* USB description */
    void* aux; /* for the device driver */
    void (*free)(void*); /* idem. to release aux */
};
struct Usbdev {
    ulong csp; /* USB class/subclass/proto */
    int vid; /* vendor id */
    int did; /* product (device) id */
    int dno; /* device release number */
    char* vendor;
    char* product;
    char* serial;
    int ls; /* low speed */
    int class; /* from descriptor */
    int nconf; /* from descriptor */
    Conf* conf[Nconf]; /* configurations */
    Ep* ep[Nep]; /* all endpoints in device */
    Desc* ddesc[Nddesc]; /* (raw) device specific descriptors */
};
struct Ep {
    uchar addr; /* endpt address */
    uchar dir; /* direction, Ein/Eout */
    uchar type; /* Econtrol, Eiso, Ebulk, Eintr */
    uchar isotype; /* Eunknown, Easync, Eadapt, Esync */
    int id;
    int maxpkt; /* max. packet size */
    Conf* conf; /* the endpoint belongs to */
    Iface* iface; /* the endpoint belongs to */
};
struct Altc {
    int attrib;
    int interval;
    void* aux; /* for the driver program */
};
struct Iface {
    int id; /* interface number */
    ulong csp; /* USB class/subclass/proto */
    Altc* altc[Naltc];
    Ep* ep[Nep];
    void* aux; /* for the driver program */
};
struct Conf {
    int cval;   /* value for set configuration */
    int attrib;
    int milliamps; /* maximum power in this config. */
    Iface* iface[Niface]; /* up to 16 interfaces */
};

struct Desc {
    Conf* conf; /* where this descriptor was read */
    Iface* iface; /* last iface before desc in conf. */
    Ep* ep; /* last endpt before desc in conf. */
    Altc* altc; /* last alt.c. before desc in conf. */
    DDesc data; /* unparsed standard USB descriptor */
};

struct DDesc {
    uchar bLength;
    uchar bDescriptorType;
    uchar bbytes[1];
    /* extra bytes allocated here to keep the rest of it */
};

#define Class(csp) ((csp)&0xff)
#define Subclass(csp) (((csp)>>8)&0xff)
#define Proto(csp) (((csp)>>16)&0xff)
#define CSP(c, s, p) ((c) | ((s)<<8) | ((p)<<16))
#define GET2(p) ...
#define PUT2(p,v) ...
#define GET4(p) ...
#define PUT4(p,v) ...
#define dprint if(usbdebug)fprint
#define ddprint if(usbdebug > 1)fprint

DESCRIPTION
This library provides convenience structures and functions to write USB device drivers. It is not intended for user programs using USB devices. See usb(3) for a description of the interfaces provided for that purpose. For drivers that provide a file system and may be embedded into usbd, the library includes a file system implementation toolkit described in usbfso(2).

Usb drivers rely on usb(3) to perform I/O through USB as well as on usbd(4) to perform the initial configuration for the device’s setup endpoint. The rest of the work is up to the driver and is where this library may help.
In most cases, a driver locates the devices of interest and configures them by calling `startdevs` and then sets up additional endpoints as needed (by calling `opendev`) to finally perform I/O by reading and writing the data files for the endpoints.

An endpoint as provided by `usb(3)` is represented by a `Dev` data structure. The setup endpoint for a device represents the USB device, because it is the means to configure and operate the device. This structure is reference counted. Functions creating `Dev`s adjust the number of references to one, initially. The driver is free to call `inref` (in `lock(2)`) to add references and `closedev` to drop references (and release resources when the last one vanishes). As an aid to the driver, the field `aux` may keep driver-specific data and the function `free` will be called (if not null) to release the `aux` structure when the reference count goes down to zero.

`Dev.dir` holds the path for the endpoint’s directory.

The field `id` keeps the device number for setup endpoints and the endpoint number for all other endpoints. For example, it would be 3 for `/dev/usb/ep3.0` and 1 for `/dev/usb/ep3.1`. It is easy to remember this because the former is created to operate on the device, while the later has been created as a particular endpoint to perform I/O.

Fields `dfd` and `cfd` keep the data and control file descriptors, respectively. When a `Dev` is created the control file is open, initially. Opening the data file requires calling `opendevdata` with the appropriate mode.

When the device configuration information has been loaded (see below), `maxpkt` holds the maximum packet size (in bytes) for the endpoint and `usb` keeps the rest of the USB information.

Most of the information in `usb` comes from parsing various device and configuration descriptors provided by the device, by calling one of the functions described later. Only descriptors unknown to the library are kept unparsed at `usb.ddesc` as an aid for the driver (which should know how to parse them and what to do with the information).

**Configuration**

`Startdevs` is a wrapper that locates devices of interest, loads their configuration information, and starts a `thread(2)`’s `proc` for each device located so that it executes `f` as its main entry point. The entry point is called with a pointer to the `Dev` for the device it has to process, `argv`, and `argc`. Devices are located either from the arguments (after options) in `argv`, if any, or by calling the helper function `mf` with the argument `ma` to determine (for each device available) if the device belongs to the driver or not. If the function returns −1 then the device is not for us.

In many cases, `matchdevcsp` may be supplied as `mf` along with a (null terminated) vector of CSP values supplied as `ma`. This function returns 0 for any device with a CSP matching one in the vector supplied as an argument and −1 otherwise. In other cases (e.g., when a particular vendor and device ids are the ones identifying the device) the driver must include its own function and supply it as an argument to `startdevs`. The first argument of the function corresponds to the information known about the device (the second line in its `ct1` file). `Openep` creates the endpoint number `id` for the device `d` and returns a `Dev` structure to operate on it (with just the control file open).

`Opendedev` creates a `Dev` for the endpoint with directory `fn`. Usually, the endpoint is a setup endpoint representing a device. The endpoint control file is open, but the data file is not. The USB description is void. In most cases drivers call `startdevs` and `opendev` and do not call this function directly.

`Configdev` opens the data file for the device supplied and loads and parses its configuration information. After calling it, the device is ready for I/O and the USB description in `Dev.usb` is valid. When using `startdevs` it is not desirable to call this function (because `startdevs` already calls it).

Control requests for an endpoint may be written by calling `devctl` in the style of `print(2)`. It is better not to call `print` directly because the control request should be issued as a single `write` system call. See `usb(3)` for a list of available control requests (not to be confused with USB control transfers performed on a control endpoint).

**Input/Output**

`Opendedevdata` opens the data file for the device according to the given `mode`. The mode must match that of the endpoint, doing otherwise is considered an error. Actual I/O is performed by reading/writing the descriptor kept in the `dfd` field of `Dev`. 
For control endpoints, it is not necessary to call `read` and `write` directly. Instead, `usbcmd` issues a USB control request to the device *d* (not to be confused with a `usb(3)` control request sent to its control file). `Usbcmd` retries the control request several times upon failure because some devices require it. The format of requests is fixed per the USB standard: `type` is the type of request and `req` identifies the request. Arguments `value` and `index` are parameters to the request and the last two arguments, `data` and `count`, are similar to `read` and `write` arguments. However, `data` may be `nil` if no transfer (other than the control request) has to take place. The library header file includes numerous symbols defined to help writing the type and arguments for a request.

The return value from `usbcmd` is the number of bytes transferred, zero to indicate a stall and −1 to indicate an error.

A common request is to uninstall an endpoint that has been stalled due to some reason by the device (eg., when read or write indicate a count of zero bytes read or written on the endpoint). The function `uninstall` does this. It is given the device that stalled the endpoint, `dev`, the stalled endpoint, `ep`, and the direction of the stall (one of `Ein` or `Eout`). The function takes care of notifying the device of the uninstall as well as notifying the kernel.

**Tools**

`Class` returns the class part of the number given, representing a CSP. `Subclass` does the same for the device subclass and `Proto` for the protocol. The counterpart is `CSP`, which builds a CSP from the device class, subclass, and protocol. For some classes, `classname` knows the name (for those with constants in the library header file).

The macros `GET2` and `PUT2` get and put a (little-endian) two-byte value and are useful to parse descriptors and replies for control requests.

Functions `emallocz` and `estdup` are similar to `mallocz` and `strdup` but abort program operation upon failure.

The function `Ufmt` is a format routine suitable for `fmtinstall(2)` to print a Dev data structure. The auxiliary `hexstr` returns a string representing a dump (in hexadecimal) of *n* bytes starting at *a*. The string is allocated using `malloc(2)` and memory must be released by the caller.

`Loaddevstr` returns the string obtained by reading the device string descriptor number `sid`.

**SOURCE**

```
/sys/src/cmd/usb/lib
```

**SEE ALSO**

`usbfs(2), usb(3), usb(4), usb(4)`.

**BUGS**

Not heavily exercised yet.
NAME

usbreadbuf, usbfsadd, usbfsdel, usbdirdread, usbfsinit, usbdirsfs, usbfs - USB device driver file system library

SYNOPSIS

```
#include <u.h>
#include <libc.h>
#include <thread.h>
#include "../lib/usb.h"
#include "../lib/usbfs.h"
enum {
    Hdrsize = 128, /* plenty of room for headers */
    Msgsize = 8 * 1024,
    Bufsize = Hdrsize + Msgsize,
    Namesz = 40,
    Errmax = 128,
    ONONE = ~0, /* omode in Fid when not open */
};
struct Fid {
    int fid;
    Qid qid;
    int omode;
    Fid* next;
    void* aux;
};
struct Usbfs {
    char name[Namesz];
    uvlong qid;
    Dev* dev;
    void* aux;
    int (*walk)(Usbfs *fs, Fid *f, char *name);
    void (*clone)(Usbfs *fs, Fid *of, Fid *nf);
    void (*clunk)(Usbfs *fs, Fid *f);
    int (*open)(Usbfs *fs, Fid *f, int mode);
    long (*read)(Usbfs *fs, Fid *f, int mode);
    long (*write)(Usbfs *fs, Fid *f, int mode);
    long (*stat)(Usbfs *fs, Qid q, Dir *d);
    void (*end)(Usbfs *fs);
};
typedef int (*Dirgen)(Usbfs*, Qid, int, Dir*, void*);
long usbreadbuf(void *data, long count, vlong offset, void *buf, long n);
void usbfsadd(Usbfs *dfs);
void usbfsdel(Usbfs *dfs);
int usbdirdread(Usbfs* f, Qid q, char *data, long cnt, vlong off, Dirgen gen, void *arg);
void usbfsinit(char* srv, char *mnt, Usbfs *f, int flag);
void usbdirdump(void);
extern char Enotfound[], Etoosmall[], Eio[], Eperm[], Ebadcall[],
       Ebadfid[], Einuse[], Eisopen[], Ebadctl[];
extern Usbfs usbdirsfs;
extern int usbfsdebug;
```

DESCRIPTION

This library provides an alternative to 9p(2) for implementing a file server within a USB driver.
Drivers using this library may be embedded into *usbd*(4). It may be also desirable to use this library when drivers are not embedded because it is tailored to work well with the library for handling USB devices.

A USB file system is described by a *Usbfs* structure. In most cases, the driver is not responsible for the root of the file tree. It is customary that a driver creates a file server for each device handled and links all of them to a root directory implemented by the *usbdifs* file system implemented by the library. This root directory is bound to /dev in most cases.

*Usbdifs* implements a root directory populated by named file trees, each one described by a *Usbfs* structure.

The field *Usbfs*.name contains the name for the root directory of the file system, usually a directory seen at /dev/name when the driver is embedded.

*Usbfs*.qid maintains a value used to decorate qids for the file tree. This may be ignored when *usbdifs* is not used. Otherwise, *usbdifs* assigns a unique value kept at the high 32 bits of *Qid*.path for all files on each file tree bound to it. Each *Usbfs* server must bitwise OR *Usbfs*.qid to all *Qid*.path values returned by its functions. In the same way, functions usually clear bits in *Usbfs*.qid before processing *Qid*.path values supplied as input.

The USB device handled by a file tree is referenced from *Usbfs*.dev (and a reference must be counted for it). This permits the following functions to quickly locate the device of interest, and also permits releasing the device when no request is outstanding.

The field *Usbfs*.aux is for the device to use. The rest of the fields implement the 9P protocol for the device. Not all the operations need be implemented. Only *walk*, *open*, *read*, *write*, and *stat*, must be implemented (and their corresponding fields in *Usbfs* may never be nil). These functions must return −1 upon failure and set the error string to reflect the cause of a failure.

In all the functions, a 9P fid is represented by a *Fid* structure. It contains the 9P *fid*, the corresponding *qid*, and an auxiliary pointer for the driver to use. Open *fids* have a valid open mode in *omode* while others have ONONE to indicate that the *fid* is not open. The library takes care of which fids exist and which ones do not.

*Walk* must walk *f* to *name* (a single name, not a file path) in the supplied *fs*. Its implementation should update the qid in *f* to reflect the walk. This function must bitwise OR any returned Qid with *Usbfs*.qid, if *usbdifs* is used.

*Clone* must clone fid of onto *nf* so that, upon successful completion, *nf* also refers to the file that *f* refers to. An implementation must update the Qid of the cloned fid. If this function is not supplied, the library copies the aux field to the cloned fid.

*Clunk* clunks *f*. It usually releases data kept in the aux field, but may be set to nil otherwise.

*Open* prepares the fid *f* for I/O according to *mode*. The open mode in the fid is updated by the library upon return. The library checks trivial cases like opening already–open fids. The implementation performs most permission checking.

*Read* reads up to *count* bytes into *data* starting at *offset* in the file referenced by *f*. *Write* is the counterpart. To read from directories, the function *usbdirread* may be called. It returns the return value of *read* or −1. *usbdirread* calls *gen* to iterate through files as needed. The Dir*gen* function will be called with index values of 0 and up to ask for the first file and following files. To read from data already in buffers, the function *usbreadbuf* may help. It must be given the arguments supplied by the user, plus the buffer and buffer size.

*Stat* must fill *d* with the directory entry for the file identified by *q*. As an aid, *d* is initialized to fake access and modification times, and user and group ids. Also, the field name in *d* is initialized to point to a 40–byte buffer. If the file name fits, it may be copied directly into d->name without allocating memory for that purpose. Otherwise d->name must be initialized to point to static memory.

The function *end* is called upon termination of the file tree to release resources.

Calling *usbfsinit* starts a file server for *f* that mounts itself at *mnt* and posts *srv* at *srv*(3). In most cases, the file system supplied is *usbdifs*. The flag is used for *mount* (see *bind*(2)). Once *usbdifs* is started, calls to *usbfsadd* add a file tree implemented by *dfs* to the root directory of *usbdifs* and calls to *usbfsdel* remove that binding (and release resources including the reference to the USB
device).

Various error strings are declared as an aid. The global usbfsdebug may be set to trigger diagnostics and protocol tracing.

**EXAMPLE**

See /sys/src/cmd/usb/disk for an example driver that uses this library. Looking at an example is strongly suggested to see how reference counts for the USB device and the file system are handled.

**SOURCE**

/sys/src/cmd/usb/lib

**SEE ALSO**

usb(2), usb(3), usb(4), usbd(4)
NAME
venti – archival storage server

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <venti.h>

DESCRIPTION
The Venti library provides support for writing Venti servers and clients. Other manual pages describe the library functions in detail.

Venti-cache(2) describes a simple in-memory block cache to help clients.
Venti-conn(2) describes routines for manipulating network connections between Venti clients and servers. Venti-client(2) and venti-server(2) describe routines for writing clients and servers on top of these.
Venti-fcall(2) describes the C representation of Venti protocol messages and data structures. It also describes routines that convert between the C representation and the network and disk representations.
Venti-file(2) describes routines for writing clients that manipulate Venti file trees (see venti(6)).
Venti-log(2) describes routines to access in-memory log buffers as well as the logging that is done automatically by the library.
Venti-mem(2) describes wrappers around the canonical malloc(2) routines that abort on error.
Venti-packet(2) describes routines for manipulating zero-copy chains of data buffers.
Venti-zero(2) describes routines to zero truncate and zero extend blocks (see venti(6)).

SOURCE
/sys/src/libventi

SEE ALSO
venti(1), venti-cache(2), venti-client(2), venti-fcall(2), venti-file(2) venti-log(2), venti-mem(2), venti-packet(2), venti-server(2), venti-zero(2), venti(6), venti(8)
NAME
VtBlock, VtCache, vtblockcopy, vtblockdirty, vtblockduplock, vtblockput, vtblockwrite, vtcachealloc,
vtcacheallocblock, vtcachefree, vtcacheglobal, vtcachelocal, vtcachesetwrite,
vtglobaltolocal, vtlotalglocal - Venti block cache

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <venti.h>
typedef struct VtBlock
{
    uchar *data;
    uchar type;
    uchar score[VtScoreSize];
    u32int addr;
    ...
} VtBlock;
VtCache* vtcachealloc(VtConn *z, int blocksize, ulong nblocks);
void vtcachefree(VtCache *c);
u32int vtcacheclocksize(VtCache *c);
u32int vtlotalglocal(uchar score[VtScoreSize])
void vtlotalglocal(u32int local, uchar score[VtScoreSize])
VtBlock* vtcacheallocblock(VtCache *c, int type);
VtBlock* vtcachelocal(VtCache *c, u32int addr, int type);
VtBlock* vtcacheglobal(VtCache *c, uchar[VtScoreSize], int type);
void vtblockput(VtBlock *b);
void vtblockduplock(VtBlock *b);
int vtblockwrite(VtBlock *b);
void vtcachesetwrite(VtCache *c,
    int (*write)(VtConn*, uchar[VtScoreSize], uint, uchar*, int));
VtBlock* vtblockcopy(VtBlock *b);
int vtblockdirty(VtBlock *b);

DESCRIPTION
These functions provide access to a simple in–memory cache of blocks already stored on a Venti
server and blocks that will eventually be stored on a Venti server.
A VtBlock represents a venti data block. Blocks stored on a venti server, called global blocks,
are named by the SHA1 hash of their contents. This hash is recorded as the block’s score. Such
blocks are immutable. The cache also stores mutable blocks that have not yet been written to a
venti server. These blocks are called local blocks, and have special scores that are 16 zero bytes
followed by a 4–byte big–endian address. The address is an index into the internal set of cache
blocks.

The user–visible contents of a VtBlock are data, a pointer to the data; type, the venti block
type; score, the block’s score; and addr, the block’s cache address.

Vtcachealloc allocates a new cache using the client connection z (see venti–conn(2) and venti–
client(2)), with room for nblocks of maximum block size blocksize.

Vtcachefree frees a cache and all the associated blocks.
Vtcacheclocksize returns the cache’s maximum block size.

Vtglobaltolocal returns the local address corresponding to the given local score. If passed a global
score, vtlotalglocal returns the special constant NIlBlock (–0). Vtlotalglocal is the oppo-
site, setting score to the local score for the cache address local.
Vtcacheallocblock allocates a new local block with the given type.
Vtcachelocal retrieves the local block at address addr from the cache. The given type must match the type of the block found at addr.
Vtcacheglobal retrieves the block with the given score and dtype from the cache, consulting the Venti server if necessary. If passed a local score, vtcacheglobal invokes vtcachelocal appropriately. The block references returned by vtcacheallocblock, vtcachelocal, and vtcacheglobal must be released when no longer needed. Vtblockput releases such a reference.
It is occasionally convenient to have multiple variables refer to the same block. Vtblockduplock increments the block’s reference count so that an extra vtblockput will be required in order to release the block.
Vtblockwrite writes a local block to the Venti server, changing the block to a global block. It calls the cache’s write function to write the block to the server. The default write function is vtwrite (see venti−client(2)); vtsetcachewrite sets it. Vtsetcachewrite is used by clients to install replacement functions that run writes in the background or perform other additional processing.
Vtblockcopy copies a block in preparation for modifying its contents. The old block may be a local or global block, but the new block will be a local block.
The cache only evicts global blocks. Local blocks can only leave the cache via vtblockwrite, which turns them into global blocks, making them candidates for eviction.
If a new cache block must be allocated (for vtcacheallocblock, vtcachelocal, vtcacheglobal, or vtblockcopy), but the cache is filled (with local blocks and blocks that have not yet been released with vtblockput), the library prints the score and reference count of every block in the cache and then aborts. A full cache indicates either that the cache is too small, or, more commonly, that cache blocks are being leaked.

SOURCE
/sys/src/libventi

SEE ALSO
venti(2), venti−client(2), venti−conn(2), venti−file(2), venti(6)
NAME

vtconnect, vthello, vtread, vttwrite, vtreadpacket, vttwritepacket, vtsync, vtping, vtrpc, ventidoublechecksha1 - Venti client

SYNOPSIS

#include <u.h>
#include <libc.h>
#include <venti.h>

Packet* vtrpc(VtConn *z, Packet *p)
int vthello(VtConn *z)
int vtconnect(VtConn *z)
int vtread(VtConn *z, uchar score[VtScoreSize],
          uint type, uchar *buf, int n)
int vttwrite(VtConn *z, uchar score[VtScoreSize],
             uint type, uchar *buf, int n)
Packet* vtreadpacket(VtConn *z, uchar score[VtScoreSize],
                      uint type, int n)
int vttwritepacket(VtConn *z, uchar score[VtScoreSize],
                   uint type, Packet *p)
int vtsync(VtConn *z)
int vtping(VtConn *z)
extern int ventidoublechecksha1; /* default 1 */

DESCRIPTION

These routines execute the client side of the venti(6) protocol.

Vtrpc executes a single Venti RPC transaction, sending the request packet p and then waiting for
and returning the response packet. Vtrpc will set the tag in the packet. Vtrpc frees p, even on
error. Vtrpc is typically called only indirectly, via the functions below.

Vthello executes a hello transaction, setting z->sid to the name used by the server. Vthello is
typically called only indirectly, via vtconnect.

Vtconnect calls vttversion (see venti−conn(2)) and vthello, in that order, returning success only if
both succeed. This sequence (calling vttversion and then vthello) must be done before the functions
below can be called.

Vtread reads the block with the given score and type from the server, stores the returned data in
memory at buf, and returns the number of bytes read. If the server’s block has size larger than n,
vttread does not modify buf and returns an error.

Vttwrite writes the n bytes in buf as a block of the given type, setting score.

Vtreadpacket and vttwritepacket are like vtread and vttwrite but return or accept the block contents
in the form of a Packet. They avoid making a copy of the data.

Vtsync causes the server to flush all pending write requests to disk before returning.

Vtping executes a ping transaction with the server.

By default, vtread and vtreadpacket check that the SHA1 hash of the returned data matches the
requested score, and vttwrite and vttwritepacket check that the returned score matches the SHA1
hash of the written data. Setting ventidoublechecksha1 to zero disables these extra checks, mainly
for benchmarking purposes. Doing so in production code is not recommended.

These functions can be called from multiple threads or procs simultaneously to issue requests in
parallel. Programs that issue requests from multiple threads in the same proc should start separate
procs running vtsendproc and vtrcvproc as described in venti−conn(2).

SOURCE

/sys/src/libventi
SEE ALSO
"venti(2), venti−conn(2), venti−packet(2), venti(6)"

DIAGNOSTICS
"Vtrpc and vtpacket return nil on error. The other routines return −1 on error.
Vtwrite returns 0 on success: there are no partial writes."
NAME
VtConn, vtconn, vtdial, vtfreeconn, vtrecv, vtversion, vtdebug, vthangup – Venti network connections

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <venti.h>
typedef struct VtConn {
  int debug;
  char *version;
  char *uid;
  char *sid;
  char addr[256];
...}
VtConn;
VtConn* vtconn(int infd, int outfd)
VtConn* vtdial(char *addr)
int vtversion(VtConn *z)
int vtrecv(VtConn *z, Packet *p)
Packet* vtrecvproc(void *z)
void vtrecvproc(void *z)
void vtdebug(VtConn *z, char *fmt, ...)
void vthangup(VtConn *z)
void vtfreeconn(VtConn *z)
extern int chattyventi;/* default 0 */

DESCRIPTION
A VtConn structure represents a connection to a Venti server (when used by a client) or to a client
(when used by a server). It contains the following user-visible fields: debug, a flag enabling
debugging prints; version, the protocol version in use; uid, the (unverified) name of the client;
sid, the (unverified) name of the server; and addr, the network address of the remote side.

Vtconn initializes a new connection structure using file descriptors infd and outfd (which may be
the same) for reading and writing. Vtdial dials the given network address (see dial(2)) and returns
a corresponding connection. It returns nil if the connection cannot be established.

Vtversion exchanges version information with the remote side as described in venti(6). The negoti-
ated version is stored in z->version.

Vtsend writes a packet (see venti–packet(2)) on the connection z. The packet p should be a formatted
Venti message as might be returned by vtcallpack; vtrecv will add the two–byte length field
(see venti(6)) at the beginning. Vtsend frees p, even on error.

Vtrecv reads a packet from the connection z. Analogous to vtsend, the data read from the connec-
tion must start with a two–byte length, but the returned packet will omit them.

By default, vtsend and vtrecv block until the packet can be written or read from the network. In a
threaded program (see thread(2)), this may not be desirable. If the caller arranges for vtsendproc
and vtrecvproc to run in their own procs (typically by calling proccreate), then vtsend and vtrecv
will yield the proc in which they are run to other threads when waiting on the network. The
void* argument to vtsendproc and vtrecvproc must be the connection structure z.

Vtdump prints the formatted message to standard error when z->debug is set. Otherwise it is a
no-op.

Vthangup hangs up a connection. It closes the associated file descriptors and shuts down send
and receive procs if they have been started. Future calls to vtrecv or vtsend will return errors.
Additional calls to \textit{vthangup} will have no effect.
\textit{Vtfreeconn} frees the connection structure, hanging it up first if necessary.
If the global variable \textit{chattyventi} is set, the library prints all Venti RPCs to standard error as they are sent or received.

\textbf{SOURCE}
\texttt{/sys/src/libventi}

\textbf{SEE ALSO}
\textit{venti(1)}, \textit{venti(2)}, \textit{venti–client(2)}, \textit{venti–packet(2)}, \textit{venti–server(2)}, \textit{venti(6)}

\textbf{DIAGNOSTICS}
Routines that return pointers return \texttt{nil} on error. Routines returning integers return 0 on success, −1 on error. All routines set \texttt{errstr} on error.
NAME
VtEntry, VtFcall, VtRoot, vtentrypack, vtentryunpack, vtcallclear, vtcallfmt, vtcallpack, vtcallunpack, vtfromdisktype, vttodisktype, vtgetstring, vtputstring, vtrootpack, vtrootunpack, vtparseescore, vtscorefmt – venti data formats

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <venti.h>
enum
{
    VtEntrySize = 40,
    VtRootSize = 300,
    VtScoreSize = 20,
};
typedef struct VtEntry
{
    ulong gen; /* generation number */
    ushort psize; /* pointer block size */
    ushort dsize; /* data block size */
    uchar type;
    uchar flags;
    uvlong size;
    uchar score[VtScoreSize];
} VtEntry;
typedef struct VtRoot
{
    char name[128];
    char type[128];
    uchar score[VtScoreSize]; /* to a Dir block */
    ushort blocksize; /* maximum block size */
    uchar prev[VtScoreSize]; /* previous root block */
} VtRoot;
void vtentrypack(VtEntry *e, uchar *buf, int index)
int vtentryunpack(VtEntry *e, uchar *buf, int index)
Packet* vtfcallpack(VtFcall *f)
int vtcallunpack(VtFcall *f, Packet *p)
void vtcallclear(VtFcall *f)
uint vttodisktype(uint type)
uint vtfromdisktype(uint type)
int vtputstring(Packet *p, char *s)
int vtgetstring(Packet *p, char **s)
void vtrootpack(VtRoot *r, uchar *buf)
int vtrootunpack(VtRoot *r, uchar *buf)
int vtparseescore(char *s, char **prefix, uchar score[VtScoreSize])
int vtcallfmt(Fmt *fmt)
int vtscorefmt(Fmt *fmt)

DESCRIPTION
These routines convert between C representations of Venti structures and serialized representations used on disk and on the network.

Vtentrypack converts a VtEntry structure describing a Venti file (see venti(6)) into a 40-byte (VtEntrySize) structure at buf+index*40. Vtentryunpack does the reverse conversion.
Vtfcallpack converts a VtFcall structure describing a Venti protocol message (see venti(6)) into a packet. Vtfcallunpack does the reverse conversion.

The fields in a VtFcall are named after the protocol fields described in venti(6), except that the type field is renamed blocktype. The msgtype field holds the one-byte message type: VtHello, VtHello, and so on.

Vtfcallclear frees the strings f->error, f->version, f->uid, f->sid, the buffers f->crypto and f->codec, and the packet f->data.

The block type enumeration defined in <venti.h> (presented in venti(6)) differs from the one used on disk and in the network protocol. The disk and network representation uses different constants and does not distinguish between VtDataType+n and VtDirType+n blocks. Vttodisktype converts a <venti.h> enumeration value to the disk value; vffromdisktype converts a disk value to the enumeration value, always using the VtDirType pointers. The VtFcall field blocktype is an enumeration value (vtfcallpack and vtfcallunpack convert to and from the disk values used in packets automatically), so most programs will not need to call these functions.

Vtputstring appends the Venti protocol representation of the string s to the packet p. Vtgetstring reads a string from the packet, returning a pointer to a copy of the string in *s. The copy must be freed by the caller. These functions are used by vtfcallpack and vtfcallunpack; most programs will not need to call them directly.

Vtrootpack converts a VtRoot structure describing a Venti file tree into the 300-byte (VtRootSize) buffer pointed to by buf. Vtrootunpack does the reverse conversion.

Vtparsescore parses the 40-digit hexadecimal string s, writing its value into score. If the hexadecimal string is prefixed with a text label followed by a colon, a copy of that label is returned in *prefix. If prefix is nil, the label is ignored.

Vtfcallfmt and vtscorefmt are print(2) formatters to print VtFcall structures and scores. Vtfcallfmt assumes that vtscorefmt is installed as %V.

SOURCE
/sys/src/libventi

SEE ALSO
venti(1), venti(2), venti(6)

DIAGNOSTICS
Vtentrypack, vtfcallpack, vtrootpack, and vtfcallclear cannot fail.

Vtentryunpack, vtrootunpack, vtputstring, vtgetstring, and vtparsescore return 0 on success, -1 on error.

Vtfcallpack returns a packet on success, nil on error.

Vttodisktype and vffromdisktype return VtCorruptType (255) when presented with invalid input.
NAME
VtFile, vtfileblock, vtfileblockscore, vtfileclose, vtfilecreate, vtfilecreateroot, vtfileflush, vtfileflushbefore, vtfilegetdirsize, vtfilegetentry, vtfilegetsize, vtfileinccref, vtfilelock, vtfilelock2, vtfileopen, vtfileopenroot, vtfileread, vtfileremove, vtfilesedirsize, vtfilesetentry, vtfilesetsize, vtfiletruncate, vtfileunlock, vtfilewrite – Venti files

SYNOPSIS
VtFile* vtfilecreateroot(VtCache *c, int psize, int dsize, int type);
VtFile* vtfileopenroot(VtCache *c, VtEntry *e);
VtFile* vtfileopen(VtFile *f, u32int n, int mode);
VtFile* vtfilecreate(VtFile *f, int psize, int dsize, int type);
void vtfileinccref(VtFile *f);
void vtfileclose(VtFile *f);
int vtfileremove(VtFile *f);
VtBlock* vtfileblock(VtFile *f, u32int n, int mode);
long vtfileread(VtFile *f, void *buf, long n, vlong offset);
long vtfilewrite(VtFile *f, void *buf, long n, vlong offset);
int vtfileflush(VtFile *f);
int vtfileflushbefore(VtFile *f, vlong offset);
int vtfiletruncate(VtFile *f);
uvlong vtfilesedirsize(VtFile *f);
int vtfilesedirsize(VtFile *f, u32int size);
int vtfilegetdirsize(VtFile *f);
int vtfilegetdirsize(VtFile *f, u32int size);
int vtfilegetentry(VtFile *f, VtEntry *e);
int vtfilegetentry(VtFile *f, VtEntry *e);
int vtfileblockscore(VtFile *f, u32int n, uchar score[VTscoreSize]);
int vtfilelock(VtFile *f, int mode);
int vtfilelock2(VtFile *f, VtFile *f, int mode);
void vtfileunlock(VtFile *f);

DESCRIPTION
These routines provide a simple interface to create and manipulate Venti file trees (see venti(6)).

Vtfilecreateroot creates a new Venti file. Type must be either VtDataType or VtDirType, specifying a data or directory file. Dsize is the block size to use for leaf (data or directory) blocks in the hash tree; psize is the block size to use for internal (pointer) blocks.

Vtfileopenroot opens an existing Venti file described by e.

Vtfileopen opens the Venti file described by the nth entry in the directory f. Mode should be one of VtOREAD, VtOWRITE, or VtORDWR, indicating how the returned file is to be used. The VtOWRITE and VtORDWR modes can only be used if f is open with mode VtORDWR.

Vtfilecreate creates a new file in the directory f with block type type and block sizes dsize and psize (see vtfilecreateroot above).

Each file has an associated reference count and holds a reference to its parent in the file tree. Vtfileinccref increments this reference count. Vtfileclose decrements the reference count. If there are no other references, vtfileclose releases the reference to f’s parent and then frees the in-memory structure f. The data stored in f is still accessible by reopening it.
Vtfileremove removes the file \( f \) from its parent directory. It also acts as vtfileclose, releasing the reference to \( f \) and potentially freeing the structure.

Vtfileblock returns the \( n \)th block in the file \( f \). If there are not \( n \) blocks in the file and \( \text{mode} \) is 
VtOREAD, vtfileblock returns nil. If the mode is 
VtOWRITE or VtORDWR, vtfileblock grows the file as needed and then returns the block.

Vtfileread reads at most \( n \) bytes at offset \( \text{offset} \) from \( f \) into memory at \( \text{buf} \). It returns the number of bytes read.

Vtfilewrite writes the \( n \) bytes in memory at \( \text{buf} \) into the file \( f \) at offset \( n \). It returns the number of bytes written, or \(-1\) on error. Writing fewer bytes than requested will only happen if an error is encountered.

Vtfilewrite writes to an in-memory copy of the data blocks (see venti-cache(2)) instead of writing directly to Venti. Vtfileflush writes all copied blocks associated with \( f \) to the Venti server. Vtfileflushbefore flushes only those blocks corresponding to data in the file before byte \( \text{offset} \). Loops that vtfilewrite should call vtfileflushbefore regularly to avoid filling the block cache with unwritten blocks.

Vtfiletruncates changes the file \( f \) to have zero length.

Vtfilegetsize returns the length (in bytes) of file \( f \).

Vtfilegetsig returns the length (in bytes) of file \( f \).

Vtfilegetdirsize returns the length (in directory entries) of the directory \( f \).

Vtfilesetdirsize sets the length (in directory entries) of the directory \( f \).

Vtfilegetentry fills \( e \) with an entry that can be passed to vtfileopenroot to reopen \( f \) at a later time.

Vtfilesetentry sets the entry associated with \( f \) to be \( e \).

Vtfileblockscore returns in \( \text{score} \) the score of the \( n \)th block in the file \( f \).

Venti files are locked and unlocked via vtfilelock and vtfileunlock to moderate concurrent access. Only one thread at a time—the one that has the file locked—can read or modify the file. The functions that return files (vtfilecreateroot, vtfileopenroot, vtfilecreate, and vtfileopen) return them unlocked. When files are passed to any of the functions documented in this manual page, it is the caller’s responsibility to ensure that they are already locked.

Internally, a file is locked by locking the block that contains its directory entry. When two files in the same directory both need to be locked, vtfilelock2 must be used. It locks both its arguments, taking special care not to deadlock if their entries are stored in the same directory block.

SOURCE
/sys/src/libventi/file.c

SEE ALSO
venti-cache(2), venti-conn(2), venti-client(2), venti(6)
NAME
VtLog, VtLogChunk, vtag, vtlogclose, vtlogdump, vtlognames, vtlogopen, vtlogprint, vtlogremove, vtlogopen, ventilogging - Venti logs

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <venti.h>
VtLog* vtlogopen(char *name, uint size);
void vtlogprint(VtLog *log, char *fmt, ...);
void vtlogclose(VtLog *log);
void vtlog(char *name, char *fmt, ...);
void vtlogremove(char *name);
char** vtlognames(int *n);
extern int ventilogging; /* default 0 */
extern char *VtServerLog; /* "libventi/server" */

DESCRIPTION
These routines provide an in-memory circular log structure used by the Venti library and the Venti server to record events for debugging purposes. The logs are named by UTF strings.

Vtlogopen returns a reference to the log with the given name. If a log with that name does not exist and size is non-zero, vtlogopen creates a new log capable of holding at least size bytes and returns it. Vtlogclose releases the reference returned by vtlogopen.

Vtlogdump writes to log, which must be open.

Vtlog is a convenient packaging of vtlogopen followed by vtlogprint and vtlogclose.

Vtlogremove removes the log with the given name, freeing any associated storage.

Vtlognames returns a list of the names of all the logs. The length of the list is returned in *n. The list should be freed by calling vtfree on the returned pointer. The strings in the list will be freed by this call as well. (It is an error to call vtfree on any of the strings in the list.)

Vtlogdump prints log, which must be open, to the file descriptor fd.

If ventilogging is set to zero (the default), vtlognames and vtlogdump can inspect existing logs, but vtlogopen always returns nil and vtlog is a no-op. The other functions are no-ops when passed nil log structures.

The server library (see venti−conn(2) and venti−server(2)) writes debugging information to the log named VtServerLog, which defaults to the string 'libventi/server'.

SOURCE
/sys/src/libventi

SEE ALSO
venti(2), venti(8)
NAME
vtbrk, vtmalloc, vtmallocz, vtrealloc, vtstrdup, vtfree – error-checking memory allocators

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <venti.h>
void* vtbrk(int size)
void* vtmalloc(int size)
void* vtmallocz(int size)
void* vtrealloc(void *ptr, int size)
char* vtstrdup(char *s)
void vtfree(void *ptr)

DESCRIPTION
These routines allocate and free memory. On failure, they print an error message and call sysfatal
(from perror(2)). They do not return.

Vtbrk returns a pointer to a new, permanently allocated block of at least size bytes.

Vtmalloc, vtrealloc, and vtstrdup are like malloc, realloc, and strdup, but, as noted above, do not
return on error. Vtmallocz is like vtmalloc but zeros the block before returning it. Memory allo-
cated with all four should be freed with vtfree when no longer needed.

SOURCE
/sys/src/libventi

SEE ALSO
venti(2)
NAME
Packet, packetalloc, packetappend, packetasize, packetcmp, packetconcat, packetconsume, packet-copy, packetdup, packetforeign, packetfragments, packetfree, packetheader, packetpeek, packet-prefix, packetshal, packetsize, packetsplit, packetstats, packettrailer, packettrim – zero-copy network buffers

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <venti.h>
Packet* packetalloc(void);
void packetappend(Packet *p, uchar *buf, int n)
uint packetasize(Packet *p)
int packetcmp(Packet *p, Packet *q)
void packetconcat(Packet *p, Packet *q)
int packetconsume(Packet *p, uchar *buf, int n)
int packetcopy(Packet *p, uchar *buf, int offset, int n)
Packet* packetdup(Packet *p, int offset, int n)
Packet* packetforeign(uchar *buf, int n,
    void (*free)(void *a), void *a)
int packetfragments(Packet *p, IORchunk *io, int nio,
    int offset)
void packetfree(Packet *p)
uchar* packetheader(Packet *p, int n)
uchar* packetpeek(Packet *p, uchar *buf, int offset, int n)
void packetprefix(Packet *p, uchar *buf, int n)
void packetshal(Packet *p, uchar shal[20])
uint packetsize(Packet *p)
Packet* packetsplit(Packet *p, int n)
void packetstats(void)
uchar* packettrailer(Packet *p, int n)
int packettrim(Packet *p, int offset, int n)

DESCRIPTION
A Packet is a chain of blocks of data. Each block, called a fragment, is contiguous in memory, but the entire packet may not be. This representation helps avoid unnecessary memory copies.
Packetalloc allocates an empty packet.
Packetappend appends the n bytes at buf to the end of p.
Packetasize returns the number of data bytes allocated to p. This may be larger than the number of bytes stored in p because fragments may not be filled completely.
Packetcmp compares the data sections of two packets as memcmp (see memory(2)) would.
Packetconcat removes all data from q, appending it to p.
Packetconsume removes n bytes from the beginning of p, storing them into buf.
Packetcopy copies n bytes at offset in p to buf.
Packetdup creates a new packet initialized with n bytes from offset in p.
Packetforeign allocates a packet containing ‘foreign’ data: the n bytes pointed to by buf. Once the bytes are no longer needed, they are freed by calling free(a).
Packetfragments initializes up to \textit{nio} of the \textit{io} structures with pointers to the data in \textit{p}, starting at \textit{offset}. It returns the total number of bytes represented by the returned structures. Packetfragments initializes any unused \textit{io} structures with nil pointer and zero length.

Packetfree frees the packet \textit{p}.

Packetheader returns a pointer to the first \textit{n} bytes of \textit{p}, making them contiguous in memory if necessary.

Packetpeek returns a pointer to the \textit{n} bytes at \textit{offset} in \textit{p}. If the requested bytes are already stored contiguously in memory, the returned pointer points at the internal data storage for \textit{p}. Otherwise, the bytes are copied into \textit{buf}, and packetpeek returns \textit{buf}.

Packetprefix inserts a copy of the \textit{n} bytes at \textit{buf} at the beginning of \textit{p}.

Packetsha1 computes the SHA1 hash of the data contained in \textit{p}.

Packetsize returns the length, in bytes, of the data contained in \textit{p}.

Packetsplit returns a new packet initialized with \textit{n} bytes removed from the beginning of \textit{p}.

Packetstats prints run–time statistics to standard output.

Packettrailer returns a pointer to the last \textit{n} bytes of \textit{p}, making them contiguous in memory if necessary.

Packettrim deletes all bytes from the packet \textit{p} except the \textit{n} bytes at offset \textit{offset}.

SOURCE
/sys/src/libventi

SEE ALSO
venti(2)

DIAGNOSTICS
These functions return errors only when passed invalid inputs, \textit{e.g.}, requests for data at negative offsets or beyond the end of a packet.

Functions returning pointers return nil on error; functions returning integers return \texttt{-1} on error. Most functions returning integers return \texttt{0} on success. The exceptions are packetfragments and packetcmp, whose return values are described above.

When these functions run out of memory, they print error messages and call sysfatal.
NAME
  vtsrvhello, vtlisten, vtgetreq, vtrespond – Venti server

SYNOPSIS

```c
#include <u.h>
#include <libc.h>
#include <venti.h>

typedef struct VtReq
{
  VtFcall tx;
  VtFcall rx;
  ...
} VtReq;

int vtsrvhello(VtConn *z)
VtSrv* vtlisten(char *addr)
VtReq* vtgetreq(VtSrv *srv)
void vtrespond(VtReq *req)
```

DESCRIPTION
These routines execute the server side of the venti(6) protocol.

`vtsrvhello` executes the server side of the initial `hello` transaction. It sets `z->uid` with the user name claimed by the other side. Each new connection must be initialized by running `vtversion` and then `vtsrvhello`. The framework below takes care of this detail automatically; `vtsrvhello` is provided for programs that do not use the functions below.

`vtlisten`, `vtgetreq`, and `vtrespond` provide a simple framework for writing Venti servers.

`vtlisten` announces at the network address `addr`, returning a fresh `VtSrv` structure representing the service.

`vtgetreq` waits for and returns the next read, write, sync, or ping request from any client connected to the service `srv`. Hello and goodbye messages are handled internally and not returned to the client. The interface does not distinguish between the different clients that may be connected at any given time. The request can be found in the `tx` field of the returned `VtReq`.

Once a request has been served and a response stored in `r->rx`, the server should call `vtrespond` to send the response to the client. `Vtrespond` frees the structure `r` as well as the packets `r->tx.data` and `r->rx.data`.

EXAMPLE
`/sys/src/cmd/venti` contains two simple Venti servers `ro.c` and `devnull.c` written using these routines. `Ro` is a read-only Venti proxy (it rejects write requests). `Devnull` is a dangerous write-only Venti server: it discards all blocks written to it and returns error on all reads.

SOURCE
`/sys/src/libventi`

SEE ALSO
`venti(2), venti−conn(2), venti−packet(2), venti(6), venti(8)`
NAME
vtzerotruncate, vtzeroextend, vtzeroscore - Venti block truncation

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <venti.h>

uint vtzerotruncate(int type, uchar *buf, uint size)
void vtzeroextend(int type, uchar *buf, uint size, uint newsize)
extern uchar vtzeroscore[VtScoreSize];

DESCRIPTION
These utility functions compute how to truncate or replace trailing zeros (for data blocks) or trailing zero scores (for pointer blocks) to canonicalize the blocks before storing them to Venti.

Vtzerotruncate returns the size of the size-byte buffer pointed to by buf ignoring trailing zeros or zero scores, according to the given type.

Vtzeroextend pads buf with zeros or zero scores, according to the given type, to grow it from size bytes to newsize bytes.

Vtzeroscore is the score of the zero-length block.

SOURCE
/sys/src/libventi/zero.c
/sys/src/libventi/zeroscore.c

SEE ALSO
venti(2), venti(6)
NAME
await, wait, waitpid - wait for a process to exit

SYNOPSIS
#include <u.h>
#include <libc.h>

Waitmsg* wait(void)
int waitpid(void)
int await(char *s, int n)

DESCRIPTION
Wait causes a process to wait for any child process (see fork(2)) to exit. It returns a Waitmsg holding information about the exited child. A Waitmsg has this structure:

typedef
struct Waitmsg
{
    int pid; /* of loved one */
    ulong time[3]; /* of loved one & descendants */
    char *msg;
} Waitmsg;

Pid is the child’s process id. The time array contains the time the child and its descendants spent in user code, the time spent in system calls, and the child’s elapsed real time, all in units of milliseconds. Msg contains the message that the child specified in exits(2). For a normal exit, msg[0] is zero, otherwise msg is the exit string prefixed by the process name, a blank, the process id, and a colon.

If there are no more children to wait for, wait returns immediately, with return value nil.

The Waitmsg structure is allocated by malloc(2) and should be freed after use. For programs that only need the pid of the exiting program, waitpid returns just the pid and discards the rest of the information.

The underlying system call is await, which fills in the n-byte buffer s with a textual representation of the pid, times, and exit string. There is no terminal NUL. The return value is the length, in bytes, of the data.

The buffer filled in by await may be parsed (after appending a NUL) using tokenize (see getfields(2)); the resulting fields are, in order, pid, the three times, and the exit string, which will be ‘’ for normal exit. If the representation is longer than n bytes, it is truncated but, if possible, properly formatted. The information that does not fit in the buffer is discarded, so a subsequent call to await will return the information about the next exiting child, not the remainder of the truncated message. In other words, each call to await returns the information about one child, blocking if necessary if no child has exited.

If the calling process has no living children, await and waitpid return −1.

SOURCE
/sys/src/libc/9syscall
/sys/src/libc/9sys

SEE ALSO
fork(2), exits(2), the wait file in proc(3)

DIAGNOSTICS
These routines set errstr.
NAME
Screen, allocscreen, publicscreen, freescreen, allocwindow, bottomwindow, bottomnwindows, top-
window, topnwindows, originwindow – window management

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <draw.h>
typedef
struct Screen
{
    Display *display; /* display holding data */
    int id; /* id of system-held Screen */
    Image *image; /* unused; for reference only */
    Image *fill; /* color to paint behind windows */
} Screen;
Screen* allocscreen(Image *image, Image *fill, int public)
Screen* publicscreen(Display *d, int id, ulong chan)
int freescreen(Screen *s)
Image* allocwindow(Screen *s, Rectangle r, int ref, int val)
void bottomwindow(Image *w)
void bottomnwindows(Image **wp, int nw)
void topwindow(Image *w)
void topnwindows(Image **wp, int nw)
int originwindow(Image *w, Point log, Point scr)
enum
{
    /* refresh methods */
    Refbackup= 0,
    Refnone= 1,
    Refmesg= 2
};

DESCRIPTION
Windows are represented as Images and may be treated as regular images for all drawing opera-
tions. The routines discussed here permit the creation, deletion, and shuffling of windows, facili-
ties that do not apply to regular images.

To create windows, it is first necessary to allocate a Screen data structure to gather them
together. A Screen turns an arbitrary image into something that may have windows upon it. It is
created by allocscreen, which takes an image upon which to place the windows (typically display->image), a fill image to paint the background behind all the windows on the image,
and a flag specifying whether the result should be publicly visible. If it is public, an arbitrary other
program connected to the same display may acquire a pointer to the same screen by calling
publicscreen with the Display pointer and the id of the published Screen, as well as the
expected channel descriptor, as a safety check. It will usually require some out-of-band coordina-
tion for programs to share a screen profitably. Freescreen releases a Screen, although it
may not actually disappear from view until all the windows upon it have also been deallocated.

Unlike allocwindow, allocscreen does not initialize the appearance of the Screen.

Windows are created by allocwindow, which takes a pointer to the Screen upon which to cre-
ate the window, a rectangle r defining its geometry, an integer pixel value val to color the window
initially, and a refresh method ref. The refresh methods are Refbackup, which provides back-
ing store and is the method used by rio(1) for its clients; Refnone, which provides no refresh and
is designed for temporary uses such as sweeping a display rectangle, for windows that are com-
pletely covered by other windows, and for windows that are already protected by backing store;
and Refmesg, which causes messages to be delivered to the owner of the window when it needs to be repainted. Refmesg is not fully implemented.

The result of allocwindow is an Image pointer that may be treated like any other image. In particular, it is freed by calling freeimage (see allocimage(2)). The following functions, however, apply only to windows, not regular images.

Bottomwindow pushes window w to the bottom of the stack of windows on its Screen, perhaps obscuring it. Topwindow pulls window w to the top, making it fully visible on its Screen. (This Screen may itself be within a window that is not fully visible; topwindow will not affect the stacking of this parent window.) Bottomwindows and Topwindows are analogous, but push or pull a group of nw windows listed in the array wp. The order within wp is unaffected.

Each window is created as an Image whose Rectangle r corresponds to the rectangle given to allocwindow when it was created. Thus, a newly created window w resides on its Screen->image at w->r and has internal coordinates w->r. Both these may be changed by a call to originwindow. The two Point arguments to originwindow define the upper left corner of the logical coordinate system (log) and screen position (scr). Their usage is shown in the Examples section.

Rio(1) creates its client windows with backing store, Refbackup. The graphics initialization routine, initdraw (see graphics(2)), builds a Screen upon this, and then allocates upon that another window indented to protect the border. That window is created Refnone, since the backing store created by rio protects its contents. That window is the one known in the library by the global name screen (a historic but confusing choice).

EXAMPLES
To move a window to the upper left corner of the display,
originwindow(w, w->r.min, Pt(0, 0));
To leave a window where it is on the screen but change its internal coordinate system so (0, 0) is the upper left corner of the window,
originwindow(w, Pt(0, 0), w->r.min);
After this is done, w->r is translated to the origin and there will be no way to discover the actual screen position of the window unless it is recorded separately.

SOURCE
/sys/src/libdraw

SEE ALSO
graphics(2), draw(2), cachechars(2), draw(3)

BUGS
The refresh method Refmesg should be finished.
NAME
workerdispatch, timerdispatch, timerrecall, recvt, sendt – worker thread management

SYNOPSIS
#include <u.h>
#include <libc.h>
#include <thread.h>
#include <worker.h>

typedef struct Worker Worker;
struct Worker {
    char name[64]; /* Pointer to work to do */
    Request *r;    /* Pointer to work to do */
    ulong version; /* Incremented when accepting new work */
    Channel *chan; /* for allocating work */
    Channel *event; /* for signalling worker */
};

void workerdispatch(void (*f)(Worker*, void*), void *arg)
void timerdispatch(void (*f)(Worker*, void*), void *arg, vlong when)
int timerrecall(void (*f)(Worker*, void*), void *arg)
int recvt(Channel *c, void *v, vlong when)
int sendt(Channel *c, void *v, vlong when)

DESCRIPTION
Typical Plan 9 servers create threads to handle incoming requests. The worker library allows these threads to be recycled so they can be reused for a next request.

Workerdispatch finds a waiting worker thread, or creates one if there are none and causes the thread to execute the function f in that thread. This function is called with two arguments, a pointer to the Worker structure and the argument arg provided by the caller of workerdispatch.

When the function returns, the worker thread makes itself available for the next request. Up to Nworker worker threads can be queued.

Timerdispatch calls workerdispatch in the future, at time when, which is given in nanoseconds (see nsec in time(2)). If when is in the past, the function is called immediately.

Timerrecall finds an outstanding timerdispatch event by function and argument and attempts to cancel it. If cancellation succeeds, it returns 1; if there is a race condition, or the function has already been called, it returns 0.

recvt and sendt execute send and receive operations with timeouts. The calls return 1 if the operation succeeded normally, 0 if it timed out and -1 if it was interrupted.

FILES
/sys/lib/acid/worker useful acid(1) functions for debugging worker programs.

SOURCE
/sys/src/libworker

SEE ALSO
error(2), thread(2), time(2)

DIAGNOSTICS
Uncaugth errors (see error(2)) in the function f cause the function to be exited and the worker thread to requeue itself.

BUGS
The Stack for worker threads is only Wstack bytes.

AUTHOR
Sape Mullender
NAME
intro - introduction to the Plan 9 devices

DESCRIPTION
A Plan 9 device implements a file tree for client processes. A file name beginning with a pound sign, such as #c, names the root of a file tree implemented by a particular kernel device driver identified by the character after the pound sign. Such names are usually bound to conventional locations in the name space. For example, after

```
bind("#c", "/dev", MREPL)
```

an ls(1) of /dev will list the files provided by the console device.

A kernel device driver is a server in the sense of the Plan 9 File Protocol, 9P (see Section 5), but with the messages implemented by local rather than remote procedure calls. Also, several of the messages (Nop, Session, Flush, and Error) have no subroutine equivalents.

When a system call is passed a file name beginning with # it looks at the next character, and if that is a valid device character it performs an attach(5) on the corresponding device to get a channel representing the root of that device’s file tree. If there are any characters after the device character but before the next / or end of string, those characters are passed as parameter aname to the attach. For example,

```
#12
```

identifies the number 2 IP protocol stack (see ip(3)).

Each kernel device has a conventional place at which to be bound to the name space. The SYNOPSIS sections of the following pages includes a bind command to put the device in the conventional place. Most of these binds are done automatically by init(8) using newns (see auth(2)) on the file /Lib/namespace (see namespace(6)). When typed to rc(1), the bind commands will need quotes to protect the # characters.

SEE ALSO
intro(5), intro(2)
NAME
aoe – ATA–over–Ethernet (AoE) interface

SYNOPSIS
bind -a #æ /dev
/dev/aoe/ctl
/dev/aoe/log
/dev/aoe/shelf.slot/config
/dev/aoe/shelf.slot/ctl
/dev/aoe/shelf.slot/devlink/0
...
/dev/aoe/shelf.slot/devlink/i
/dev/aoe/shelf.slot/ident
...

DESCRIPTION
The AoE (ATA–over–Ethernet) interface serves a three–level directory providing control and access
to AoE targets. The interface provided is primarily intended for low–level control of the AoE initia-
tor. See sdæoe(3) for the standard interface.

Top–level files
In order to access AoE targets, one or more Ethernet controllers need to be bound to the AoE initia-
tor. By default, the system starts with no interfaces bound. For automatic binding of interfaces on
boot, the aoeif configuration variable is set in plan9.ini(8). Ethernet interfaces are specified as
erthrn, not as #1n. To bind the first and second Ethernet devices on boot, add

```
  aoeif=ether0 ether1
```

To bind ether1 to a running system:

```
  % echo bind '#l1/ether1' >/dev/aoe/ctl
```

And to unbind it

```
  % echo unbind '#l1/ether1' >/dev/aoe/ctl
```

When an interface is unbound, targets depending on that interface are removed.

Each local interface is called a netlink. The mapping of AoE targets to netlinks is called a devlink.
Each devlink may see multiple interfaces per target. For example, if the local machine has one Eth-
ernet address bound and the target has two interfaces on the same Ethernet segment, this will
result in one netlink and one devlink with two Ethernet addresses. AoE frames are sent in round–
robin fashion. Each successive frame is sent on the next address available on the next available
devlink (local interface).

Normally the initiator automatically discovers and adds new device directories on startup. New
devices are not added except as new interfaces are bound to the initiator. Several messages can
be written to /dev/aoe/ctl which alter this behavior:

autodiscover toggle
  If toggle is absent, the state of autodiscover is toggled. If it is the string on, it is
turned on. Any other string turns autodiscover off. This option is not useful after Eth-
ernet devices have been bound.

discover shelf.slot
  Attempt to find the named target on all bound interfaces.

remove shelf.slot
  The converse of discover: remove the named target if it exists.

rediscover toggle
  Allow or disallow rediscovery. This allows for automatic discovery of new targets. Unfortu-
nately, it also allows automatic modification or loss of existing targets. This option is con-
sidered dangerous.
Reading `/dev/aoe/ctl` returns a list of colon-separated lines with keywords and their values:

- **debug**
- **autodiscover**
- **rediscover** Returns the current state of the variable named by the keyword. Writing the variable's name to the control file toggles the state of that variable.
- **ifn path** Path to nth bound Ethernet device.
- **ifn ea** Ethernet address of this device.
- **ifn flag** A flag of “Up” indicates that this interface is available.
- **ifn lostjumbo** Number of consecutive lost jumbograms.
- **ifn datamtu** Incorrect and unused.

### Shelf-and-slot subdirectories

Once configured, each AoE target is accessed via files in the directory named for its shelf and slot. For example, shelf 42, slot 0 would be accessed through the path `/dev/aoe/42:0`. The `ident` file contains the read–only, verbatim result of the identify unit ATA command. The `config` file contains the target's AoE configuration string. Writing to this file sets the targets configuration string.

Reading a shelf and slot's `ctl` file returns a list of colon-separated lines with the following keywords and values:

- **state** “Up” or “down”.
- **nopen** Number of clients using this target.
- **nout** Number of outstanding AoE frames.
- **nmmaxout** Maximum number of outstanding frames allowed.
- **nframes** Maximum number of outstanding frames. *Nframes* is greater than *nmmaxout* when the initiator is reducing the number of in-flight frames due to packet loss. It is assumed that packet loss is due to an overwhelmed target and not poor network conditions.
- **maxbcount** Maximum number of data bytes per AoE frame. Using standard frames, **maxbcount** is 1024 or two sectors. AoE ATA headers are 36 bytes.
- **model**
- **serial**
- **firmware** The respective fields from the ATA `identify unit` command.
- **flag** List of flags useful for debugging. The flag *jumbo* indicates that jumbo frames are accepted, not that they are being used. **Maxbcount** should be consulted for this purpose.

The `data` file may be read or written like a normal file except that reads and writes to this file are converted to AoE commands to the target, so transfers should be 512 or 1024 bytes long (or a larger multiple of 512 if jumbo packets are in use). The size of this file is the usable size of the target.

The `devlink` directory contains one file for each interface the target was discovered on. The files are numbers from 0 to *n* and contain a list of colon-separated lines with keywords and their values:

- **addr** A space-separated list of the target's Ethernet addresses visible from this interface.
- **npkt** The number of frames sent on this interface.
- **resent** The number of frames re-sent. Frames are re-sent when they have been outstanding twice the RTT average.
- **flag** “Up” when the netlink is up.
- **rrtavg**
- **mintimer** Minimum timer and RTT average as per *Congestion Avoidance and Control*. 
nl path Path of the Ethernet device.
nl ea Ethernet address of the local Ethernet device.
nl flag “Up” if the local interface is up.
nl lostjumbo Number of consecutive jumbograms lost.
nl datamtu Unused.

SOURCE
/sys/src/9/port/devaoe.c

SEE ALSO
sd(3), sdao(3), aoesrv(8), snoopy(8)
http://www.coraid.com/documents/AoEr10.txt

BUGS
There is no raw file for executing arbitrary commands.
This is a fairly primitive interface; sdao(3) is usually more suitable.
NAME
  apm - Advanced Power Management 1.2 BIOS interface

SYNOPSIS
  bind -a #P /dev
  /dev/apm

DESCRIPTION
  This device presents a low-level interface to the APM 1.2 bios calls. It is enabled by adding the
  line "apm0=" to plan9.ini. (The value after the equals sign is ignored; the presence of the line at
  all enables the driver.) It is only available on uniprocessor PCs. Writing a 386 Ureg structure and
  then reading it back executes an APM call: the written registers are passed to the call, and the read
  registers are those returned by the call.

  This device is intended to enable more user-friendly interfaces such as apm(8).

SOURCE
  /sys/src/9/pc/apm.c
  /sys/src/9/pc/apmjum.p
NAME
arch - architecture-specific information and control

SYNOPSIS
bind -a #P /dev
/dev/archctl
/dev/cputype
/dev/ioalloc
/dev/iob
/dev/iol
/dev/iow
/dev/irqalloc

DESCRIPTION
This device presents textual information about PC hardware and allows user-level control of the
I/O ports on x86-class and DEC Alpha machines.

Reads from cputype recover the processor type and clock rate in MHz. Reads from archctl yield at
least data of this form:

    cpu AMD64 2201 pge
    pge on
    coherence mfence
    cmpswap cmpswap486
    i8253set on
    cache default uc
    cache 0x0 1073741824 wb
    cache 0x3ff00000 1048576 uc

Where AMD64 is the processor type, 2201 is the processor speed in MHz, and pge is present only
if the 'page global extension' capability is present; the next line reflects its setting. coherence is
followed by one of mb386, mb586, mfence or nop, showing the form of memory barrier used
by the kernel. cmpswap is followed by cmpswap386 or cmpswap486, reflecting the form of
'compare and swap' used by the kernel. i8253set is a flag, indicating the need to explicitly set
the Intel 8253 or equivalent timer. There may be lines starting with cache that reflect the state of
memory caching via MTRRs (memory-type region registers). The second word on the line is
default or a C-style number which is the base physical address of the region; the third is a C-
style length of the region; and the fourth is one of uc (for uncacheable), wb (write-back), wc
(write-combining), wp (write-protected), or wt (write-through). A region may be a subset of
another region, and the smaller region takes precedence. This may be used to make I/O registers
uncacheable in the midst of a write-combining region mostly used for a video framebuffer, for
example. Control messages may be written to archctl and use the same syntax as the data read
from archctl. Known commands include cache, coherence, i8253set, and pge.

Reads from ioalloc return I/O ranges used by each device, one line per range. Each line contains
three fields separated by white space: first address in hexadecimal, last address, name of device.

Reads from irqalloc return the enabled interrupts, one line per interrupt. Each line contains three
fields separated by white space: the trap number, the IRQ it is assigned to, and the name of the
device using it.

Reads and writes to iob, iow, and iol cause 8-bit wide, 16-bit wide, and 32-bit wide requests to
I/O ports. The port accessed is determined by the byte offset of the file descriptor.

EXAMPLE
The following code reads from an x86 byte I/O port.

    uchar
    inportb(unsigned port)
    {
        uchar data;

        if(iobfd == -1)
            iobfd = open("#/P/iob", ORDWR);
seek(iobfd, port, 0);
if(read(iobfd, &data, sizeof(data)) != sizeof(data))
    sysfatal("inportb(0x%4.4ux): %r", port);
return data;

SOURCE
/sys/src/9/pc/devarch.c
NAME
  audio - SoundBlaster or ESS1688 audio controller

SYNOPSIS
  bind -a #A /dev
  /dev/audio
  /dev/volume

DESCRIPTION
  The audio device serves a one-level directory, giving access to the stereo audio ports. Audio is
  the data file, which can be read or written to use the port. Audio data is a sequence of stereo sam-
  ples, left sample first. Each sample is a 16 bit little-endian two's complement integer; the default
  sampling rate is 44.1 kHz. Some implementations only support audio output and return a zero
  length when read.

  The length of the audio file as returned by stat(2) represents the number of bytes buffered for
  input or output. This provides some control over record or playback latency.

  The file audiostream provides additional timing and latency control. When read, it returns lines of
  the form

  bufsize s buffered b offset o time t

  reporting number of bytes s used for DMA operations (i.e., the minimum useful size for reads and
  writes), the number of bytes b currently buffered, and the time t at which offset o was reached.
  Using t and o, it is possible to calculate at what time a byte with a different offset will be recorded
  or played back.

  Volume is the control file associated with the audio port. Each input and output source has an
  associated stereo volume control, ranging from 0 (quiet) to 100 (loud). In addition, there are con-
  trols for the sampling rate of the D/A and A/D converters and for any tone controls. Reads return
  lines of the form

  source in left value right value out left value right value

  possibly abbreviated if the values are shared or non-existent. For example, if all of the values are
  shared, the form degenerates to 'source value'. Valid sources depend on the particular audio
  device, though all devices have an audio stereo source, which controls the output volume from
  the D/A converter connected to audio.

  Writes accept the same format with same abbreviations. Writing the string reset sets all of the
  attributes to their default value, and if no attribute is supplied, audio is assumed.

  The Sound Blaster 16 (or MCD) is half-duplex and accepts the following controls on its volume
  file, in the format shown above for reads.

  audio out    Data written to audio.
  synth in out MIDI synthesizer.
  cd in out    CD player.
  line in out  Line-level input.
  mic in out   Monaural microphone input.
  speaker in out Monaural internal speaker connection.
  treb out     Stereo treble tone control. Values less than 50 decrease the treble, those
               greater increase it.
  bass out     Stereo bass tone control.
  speed in out Sampling rate for the D/A and A/D converters, expressed in Hz. Defaults
               to 44100.

SOURCE
  /sys/src/9/port/devaudio.c

SEE ALSO
  usb(4)
NAME
bridge - IPv4 Ethernet bridge

SYNOPSIS
bind -a #Bb /net
/net/bridge/ctl
/net/bridge/cache
/net/bridge/log
/net/bridge/stats
/net/bridge/n
/net/bridge/n/ctl
/net/bridge/n/local
/net/bridge/n/status

description
The bridge device bridges IPv4 packets amongst Ethernet interfaces. The number b in the bind is optional and selects a particular bridge (default 0).

The /net/bridge0 directory contains ctl, cache, log, and stats files, and numbered subdirectories for each physical interface.

Opening the ctl file reserves an interface. The file descriptor returned from the open(2) will point to the control file, ctl, of the newly allocated interface. Reading ctl returns a text string representing the number of the interface. Writing ctl alters aspects of the interface. The possible ctl messages are:

bind ether name ownhash path
Treat the device mounted at path (e.g., /net/ether0) as an Ethernet medium carrying IPv4 and ARP packets and associate it with this bridge (forward its packets to the other interfaces associated with this bridge). Ownhash is an 'owner hash'.

bind tunnel name ownhash path path2
Treat the device mounted at path as a network tunnel carrying IPv4 and ARP packets, the device mounted at path2 as an Ethernet medium carrying IPv4 and ARP packets and associate them with this bridge (forward its packets to the other interfaces associated with this bridge). Read packets from the path interface and write them to the path2 interface. Such tunnels have an MTU of 1400 bytes.

unbind type address [ownhash]
Disassociate the interface associated with address from this bridge. Type must be ether or tunnel.

cache flush
Clear the cache of (destination MAC address, port) tuples.

delay delay0 delayn
Set the delay0 and delayn parameters. delay0 is the constant microsecond delay per packet and delayn is the microsecond delay per byte.

set option
Set bridge option. The only known option is tcpmss, which limits the TCP Maximum Segment Size of TCPv4 packets passing through to 1300 bytes.

clear option
Clear bridge option.

Reading stats returns statistics about the bridge.

Reading the log file returns data from the bridge's log and will block at end of file awaiting new data.

Reading the cache file prints the cache of (destination MAC address, port) tuples, one entry per line. The format is: the destination MAC (e.g., Ethernet) address in hex, port number, count of packets from this address, count of packets to this address, expiry time in seconds since the epoch, and e for expired entries or v for valid entries.

In a connection subdirectory, ctl and local don't do anything, but status returns a one-line status summary.
EXAMPLES

Set up a network bridge between two Ethernets (#10 and #11).

bind -a '#B' /net
bind -a '#11' /net
echo 'bind ether outer 0 /net/ether0' > /net/bridge0/ctl
echo 'bind ether inner 0 /net/ether1' > /net/bridge0/ctl

SEE ALSO

ip(3)

SOURCE

/sys/src/9/port/devbridge.c

BUGS

Doesn’t understand IPv6.
NAME
cap - capabilities for setting the user id of processes

SYNOPSIS
bind #= dir

dir/caphash
dir/capuse

DESCRIPTION
This device enables a trusted process to create a capability that another process may then use to change its user id. The intent is to allow server processes, for example telnetd (see ipserv(8)), to change their user id after having proved to a trusted process, such as factotum(4), that they are indeed executing on behalf of a user. A trusted process is one running with the user id of the host owner (see /dev/hostowner in cons(3)).

A capability is a null terminated string consisting of the concatenation of an old user name, an "@", a new user name, an "@", and a string of randomly generated characters called the key. The trusted process enables the kernel to authenticate capabilities passed to it by writing to caphash a secure hash of the capability. The hash is 20 bytes long and generated by the following call:

hmac_sha1(old_at_new, strlen(old_at_new), key, strlen(key),
hash, nil);

The kernel maintains a list of hashes, freeing them after the corresponding capability is used or after a minute has passed since the write to caphash.

The trusted process may then pass the capability to any process running as the old user. That process may then use the capability to change identity to the new user. A process uses a capability by writing it to capuse. The kernel computes the same hash using the supplied capability and searches its list of hashes for a match. If one is found, the kernel sets the process’s user id to that in the capability.

SOURCE
/sys/src/9/port/devcap.c

SEE ALSO
sechash(2)

diagnostics
Errors generated by reading and writing caphash and capuse can be obtained using errstr(2). A read of caphash with a length of less than 20 or a write to capuse that doesn’t contain two @ characters generates the error "read or write too small". A write to capuse that has no matching hash generates the error "invalid capability".
NAME
cons - console, clocks, process/process group ids, user, null, reboot, etc.

SYNOPSIS
bind #c /dev
/dev/bintime
/dev/config
/dev/cons
/dev/consctl
/dev/cputime
/dev/drivers
/dev/hostdomain
/dev/hostowner
/dev/kmsg
/dev/kprint
/dev/null
/dev/osversion
/dev/pgrp
/dev/pid
/dev/ppid
/dev/random
/dev/reboot
/dev/swap
/dev/sysname
/dev/sysstat
/dev/time
/dev/user
/dev/zero

DESCRIPTION
The console device serves a one-level directory giving access to the console and miscellaneous information.

Reading the cons file returns characters typed on the keyboard. Normally, characters are buffered to enable erase and kill processing. A control-U, ^U, typed at the keyboard kills the current input line (removes all characters from the buffer of characters not yet read via cons), and a backspace erases the previous non-kill, non-erase character from the input buffer. Killing and erasing only delete characters back to, but not including, the last newline. Characters typed at the keyboard actually produce 16-bit runes (see utf(6)), but the runes are translated into the variable-length UTF encoding (see utf(6)) before putting them into the buffer. A read(2) of length greater than zero causes the process to wait until a newline or a ^D ends the buffer, and then returns as much of the buffer as the argument to read allows, but only up to one complete line. A terminating ^D is not put into the buffer. If part of the line remains, the next read will return bytes from that remainder and not part of any new line that has been typed since.

If the string rawon has been written to the consctl file and the file is still open, cons is in raw mode: characters are not echoed as they are typed, backspace, ^U and ^D are not treated specially, and characters are available to read as soon as they are typed. Ordinary mode is reentered when rawoff is written to consctl or this file is closed.

A write (see read(2)) to cons causes the characters to be printed on the console screen.

The osversion file contains a textual representation of the operating system’s version and parameters. At the moment, it contains one field: the 9P protocol version, currently 2000.

The config file contains a copy of the kernel configuration file used to build the kernel.

The kmsg file holds the last 16 kilobytes of output written to the console by the kernel’s print statements or by processes writing to /dev/cons. It is useful for retrieving boot messages once the boot process is over.
The kprint file may be read to receive a copy of the data written to the console by the kernel's print statements or by processes writing to /dev/cons. Only data written after the file is opened is available. If the machine's console is a serial line, the data is sent both to the console and to kprint; if its console is a graphics screen, the data is sent either to the display or to kprint, but not both. (It is advisable not to open kprint on terminals until you have started rio(1).)

The null file throws away anything written to it and always returns zero when read.

The zero file is a read–only file that produces an infinite stream of zero–valued bytes when read.

The drivers file contains, one per line, a listing of the drivers configured in the kernel, in the format

    
c  cons

The hostdomain file contains the name of the authentication domain that this host belongs to; see authsrv(6). Only the user named in /dev/hostowner may write this.

The hostowner file contains the name of the user that owns the console device files. The hostowner also has group permissions for any local devices.

Reads from random return a stream of random numbers. The numbers are generated by a low priority kernel process that loops incrementing a variable. Each clock tick the variable is sampled and, if it has changed sufficiently, the last few bits are appended to a buffer. This process is inefficient at best producing at most a few hundred bits a second. Therefore, random should be treated as a seed to pseudo–random number generators which can produce a faster rate stream.

Writing the string reboot to reboot causes the system to shutdown and, if possible, restart. Writing the string reboot kernelpath loads the named kernel image and restarts, preserving the kernel configuration in #ec, except that the bootfile variable is set to kernelpath. Only the host owner has the ability to open this file.

Bintime is a binary interface that provides the same information as time (q.v.), in binary form, and also controls clock frequency and clock trim. All integers read or written from bintime are in big endian order. Unlike the other files, reads and writes do not affect the offset. Therefore, there is no need for a seek back to zero between subsequent accesses. A read of bintime returns 24 bytes, three 8 byte numbers, representing nanoseconds since start of epoch, clock ticks, and clock frequency.

A write to bintime is a message with one of 3 formats:

n<8–byte time>  set the nanoseconds since epoch to the given time.

f<8–byte freq>  Set the frequency for interpreting clock ticks to be freq ticks per second.

The rest of the files contain (mostly) read–only strings. Each string has a fixed length: a read(2) of more than that gives a result of that fixed length (the result does not include a terminating zero byte); a read of less than that length leaves the file offset so the rest of the string (but no more) will be read the next time. To reread the file without closing it, seek must be used to reset the offset. When the file contains numeric data each number is formatted in decimal. If the binary number fits in 32 bits, it is formatted as an 11 digit decimal number with leading blanks and one trailing blank; totaling 12 bytes. Otherwise, it is formatted as 21 digit decimal numbers with leading blanks and one trailing blank; totaling 22 bytes.

The cputime file holds six 32–bit numbers, containing the time in milliseconds that the current process has spent in user mode, system calls, real elapsed time, and then the time spent, by exited children and their descendants, in user mode, system calls, and real elapsed time.

The time file holds one 32–bit number representing the seconds since start of epoch and three 64–bit numbers, representing nanoseconds since start of epoch, clock ticks, and clock frequency.

A write of a decimal number to time will set the seconds since epoch.

The sysname file holds the textual name of the machine, e.g. kremvax, if known.

The sysstat file holds 10 numbers: processor number, context switches, interrupts, system calls, page faults, TLB faults, TLB purges, load average, idle time and time spent servicing interrupts. The load average is in units of milli–CPUs and is decayed over time; idle time and interrupt
time are percentage units; the others are total counts from boot time. If the machine is a multiprocessor, sysstat holds one line per processor. Writing anything to sysstat resets all of the counts on all processors.

The swap device holds a text block giving memory usage statistics:

- n memory
- n pagesize
- n kernel
- n/m user
- n/m swap
- n/m kernel malloc
- n/m kernel draw

These are total memory (bytes), system page size (bytes), kernel memory (pages), user memory (pages), swap space (pages), kernel malloced data (bytes), and kernel graphics data (bytes). The expression n/m indicates n used out of m available. These numbers are not blank padded.

To turn on swapping, write to swap the textual file descriptor number of a file or device on which to swap. See swap(8).

The other files served by the cons device are all single numbers:

- pgrp id  process group number
- pid  process number
- ppid  parent’s process number

SEE ALSO

draw(3), keyboard(6), authsrv(6), utf(6), swap(8)

SOURCE
/sys/src/9/port/devcons.c

BUGS

For debugging, two control-T’s followed by a letter generate console output and manage debugging: ^T^Td toggles whether the console debugger will be run if the system fails. ^T^Tc starts the console debugger immediately. ^T^Tk kills the largest process; use with care. ^T^Tp prints data about processes. ^T^Tq prints the run queue for processor 0. ^T^Ts prints the kernel stack. ^T^Tt prints data about kernel memory allocation.

The system can be rebooted by typing ^T^Tr.
NAME
draw - screen graphics

SYNOPSIS
bind -a #i /dev

/dev/draw/new
/dev/draw/n/ctl
/dev/draw/n/data
/dev/draw/n/colormap
/dev/draw/n/refresh

#include <u.h>
#include <draw.h>

ushort BGSHORT(uchar *p)
ulong BGLONG(uchar *p)
void BPSHORT(uchar *p, ushort v)
void BPLONG(uchar *p, ulong v)

DESCRIPTION
The draw device serves a three-level file system providing an interface to the graphics facilities of the system. Each client of the device connects by opening /dev/draw/new and reading 12 strings, each 11 characters wide followed by a blank: the connection number (n), the image id (q.v) of the display image (always zero), the channel format of the image, the min.x, min.y, max.x, and max.y of the display image, and the min.x, min.y, max.x, and max.y of the clipping rectangle. The channel format string is described in image(6), and the other fields are decimal numbers.

The client can then open the directory /dev/draw/n/ to access the ctl, data, colormap, and refresh files associated with the connection.

Via the ctl and draw files, the draw device provides access to images and font caches in its private storage, as described in graphics(2). Each image is identified by a 4-byte integer, its id.

Reading the ctl file yields 12 strings formatted as in /dev/draw/new, but for the current image rather than the display image. The current image may be set by writing a binary image id to the ctl file.

A process can write messages to data to allocate and free images, fonts, and subfonts; read or write portions of the images; and draw line segments and character strings in the images. All graphics requests are clipped to their images. Some messages return a response to be recovered by reading the data file.

The format of messages written to data is a single letter followed by binary parameters; multibyte integers are transmitted with the low order byte first. The BPSHORT and BPLONG macros place correctly formatted two- and four-byte integers into a character buffer. BGSHORT and BGLONG retrieve values from a character buffer. Points are two four-byte numbers: x, y. Rectangles are four four-byte numbers: min x, min y, max x, and max y. Images, screens, and fonts have 32-bit identifiers. In the discussion of the protocol below, the distinction between identifier and actual image, screen, or font is not made, so that “the object id” should be interpreted as “the object with identifier id”. The definitions of constants used in the description below can be found in draw.h.

The following requests are accepted by the data file. The numbers in brackets give the length in bytes of the parameters.

Allocate a new Screen (see window(2)) with screen identifier id using backing store image imageid, filling it initially with data from image fillid. If the public byte is non-zero, the screen can be accessed from other processes using the publicscreen interface.
Allocate an image with a given id on the screen named by screenid. The image will have
rectangle r and clipping rectangle clipr. If rep is non-zero, the image’s replicate bit will
be set (see draw(2)).

Refresh specifies the method to be used to draw the window when it is uncovered.
Refbackup causes the server to maintain a backing store, Refnone does not refresh
the image, and Refmsg causes a message to be sent via the refresh file (q.v.).

The image format is described by chan, a binary version of the channel format string.
Specifically, the image format is the catenation of up to four 8-bit numbers, each describing
a particular image channel. Each of these 8-bit numbers contains a channel type in its
high nibble and a bit count in its low nibble. The channel type is one of CRed, CGreen,
CBlue, CGrey, CALpha, CMap, and CIgnore. See image(6).

Color is the catenation of four 8-bit numbers specifying the red, green, blue, and alpha
channels of the color that the new image should be initially filled with. The red channel is
in the highest 8 bits, and the alpha in the lowest. Note that color is always in this format,
independent of the image format.

Change the replicate bit and clipping rectangle of the image dstid. This overrides whatever
settings were specified in the allocate message.

Use the draw operator to combine the rectangle dstr of image dstid with a rectangle of
image srcid, using a rectangle of image maskid as an alpha mask to further control blending.
The three rectangles are congruent and aligned such that the upper left corner dstr in
image dstid corresponds to the point srcp in image srcid and the point maskp in image
maskid. See draw(2).

D debugon[1]
If debugon is non-zero, enable debugging output. If zero, disable it. The meaning of
“debugging output” is implementation dependent.

Draw an ellipse in image dst centered on the point c with horizontal and vertical semiaxes
a and b. The ellipse is drawn using the image src, with the point sp in src aligned with c in
dst. The ellipse is drawn with thickness 1 + 2×thick.

If the high bit of alpha is set, only the arc of the ellipse from degree angles alpha to phi is
drawn. For the purposes of drawing the arc, alpha is treated as a signed 31-bit number
by ignoring its high bit.

Draws an ellipse or arc as the e message, but rather than outlining it, fills the correspond-
ing sector using the image srcid. The thick field is ignored, but must be non-negative.

f id[4]
Free the resources associated with the image id.

F id[4]
Free the the screen with the specified id. Windows on the screen must be freed separately.

Treat the image id as a font cache of n character cells, each with ascent ascent.

Load a character into the font cache associated with image cacheid at cache position
index. The character data is drawn in rectangle r of the font cache image and is fetched
from the congruent rectangle in image srcid with upper left corner sp. Width specifies the
width of the character—the spacing from this character to the next—while left specifies the
horizontal distance from the left edge of the character to the left side of the cache image.
The dimensions of the image of the character are defined by r.

Draw a line of thickness 1 + 2×thick in image dstid from point p0 to pl. The line is drawn
using the image srcid, translated so that point sp in srcid aligns with p0 in dstid. The end0

594
and end1 fields specify whether the corresponding line end should be a square, a disc, or an arrow head. See line in draw(2) for more details.

If in is non–zero, associate the image id with the string name. If in is zero and name already corresponds to the image id, the association is deleted.

Introduce the identifier id to correspond to the image named by the string name.

Position the window id so that its upper left corner is at the point scr on its screen. Simultaneously change its internal (logical) coordinate system so that the point log corresponds to the upper left corner of the window.

O op[1]
Set the compositing operator to op for the next draw operation. (The default is SoverzD).

Draw a polygon of thickness 1+2×thick. It is conceptually equivalent to a series of n line–drawing messages (see L above) joining adjacent points in the list of points dp. The source image srcid is translated so that the point sp in srcid aligns with the first point in the list dp. The polygon need not be closed: end0 and end1 specify the line endings for the first and last point on the polygon. All interior lines have rounded ends to make smooth joins.

Draw a polygon as the p message, but fill it rather than outlining it. The winding rule parameter wind resolves ambiguities about what to fill if the polygon is self–intersecting. If wind is ~0, a pixel is inside the polygon if the polygon’s winding number about the point is non–zero. If wind is 1, a pixel is inside if the winding number is odd. Complementary values (0 or ~1) cause outside pixels to be filled. The meaning of other values is undefined. The polygon is closed with a line if necessary.

Cause the next read of the data file to return the image pixel data corresponding to the rectangle r in image id.

Draw in the image dstid the text string specified by the n cache indices into font fontid, starting with the upper left corner at point p in image dstid. The image drawn is taken from image srcid, translated to align sp in srcid with dp in dstid. All drawing is confined to the clipping rectangle clipr in dstid.

Like the string drawing s command, but fill the background of each character with pixels from image bgid. The image bgid is translated so that the point bp aligns with the point dp in dstid.

S id[4] chan[4] Attach to the public screen with the specified id. It is an error if the screen does not exist, is not public, or does not have the channel descriptor chan for its associated image.

Send n windows to the top (if t is non–zero) or bottom (if t is zero) of the window stack. The window is specified by the list of n image ids are moved as a group, maintaining their own order within the stack.

v
Flush changes from a soft screen, if any, to the display buffer.

Replace the rectangle r of pixels in image id with the pixel data in buf. The pixel data must be in the format dictated by id’s image channel descriptor (see image(6)). The y message uses uncompressed data, while the Y message uses compressed data. In either case, it is an error to include more data than necessary.
Reading the colormap returns the system color map used on 8-bit displays. Each color map entry consists of a single line containing four space-separated decimal strings. The first is an index into the map, and the remaining three are the red, green, and blue values associated with that index. The color map can be changed by writing entries in the above format to the colormap file. Note that changing the system color map does not change the color map used for calculations involving m8 images, which is immutable.

The refresh file is read-only. As windows owned by the client are uncovered, if they cannot be refreshed by the server (such as when they have refresh functions associated with them), a message is made available on the refresh file reporting what needs to be repainted by the client. The message has five decimal integers formatted as in the ctl message: the image id of the window and the coordinates of the rectangle that should be refreshed.

SOURCE
/sys/src/9/port/devdraw.c
/sys/src/libmemdraw

DIAGNOSTICS
Most messages to draw can return errors; these can be detected by a system call error on the write (see read(2)) of the data containing the erroneous message. The most common error is a failure to allocate because of insufficient free resources. Most other errors occur only when the protocol is mishandled by the application. Errstr(2) will report details.

BUGS
The Refmesg refresh method is not fully implemented. The colormap files only reference the system color map, and as such should be called /dev/colormap rather than /dev/draw/n/colormap.
NAME
dup – dups of open files

SYNOPSIS
bind #d /fd

/fd/0
/fd/0ctl
/fd/1
/fd/1ctl
...

DESCRIPTION
The dup device serves a one–level directory containing files whose names are decimal numbers. Each such file also has an associated control file. A file of name n corresponds to open file descriptor n in the current process.

An open(2) of file n results in a file descriptor identical to what would be returned from a system call dup(n, −1). Note that the result is no longer a file in the dup device.

The stat operation returns information about the device file, not the open file it points to. A stat of #d/n will contain n for the name, 0 for the length, and 0400, 0200, or 0600 for the mode, depending on whether the dup target is open for reading, writing, or both.

A file of name nctl may be read to discover the properties of the associated file descriptor, in format identical to that of the fd file in proc(3).

SEE ALSO
dup(2)

SOURCE
/sys/src/9/port/devdup.c
NAME
env - environment variables

SYNOPSIS
bind #e /env
/env/ var1
/env/ var2
...

DESCRIPTION
The env device serves a one-level directory containing files with arbitrary names and contents. The intention is that the file name is the name of an environment variable (see rc(1)), and the content is the variable’s current value.

When a fork(2) system call creates a new process, both the parent and the child continue to see exactly the same files in the env device: changes made in either process can be noticed by the other. In contrast, an rfork system call with the RFENVG bit set (see fork(2)) causes a split: initially both process groups see the same environment files, but any changes made in one process group cannot be noticed by the other. An rfork with RFCENVG splits and then clears the environment.

The special global environment #ec contains kernel configuration variables, such as those set in plan9.ini(8). All processes see the same #ec; its contents are writable only by the host owner. [XXX actually everything is world writable; that’s a mistake.]

SEE ALSO
rc(1), fork(2), #c/reboot in cons(3), plan9.ini(8)

SOURCE
/sys/src/9/port/deenv.c

BUGS
A write starting at an offset after the current extent of a file yields an error instead of zero filling.
NAME
ether - Ethernet device

SYNOPSIS
bind -a #1n /net
/net/ethern/clone
/net/ethern/addr
/net/ethern/ifstats
/net/ethern/stats
/net/ethern/[0-7]
/net/ethern/[0-7]/data
/net/ethern/[0-7]/ctl
/net/ethern/[0-7]/ifstats
/net/ethern/[0-7]/stats
/net/ethern/[0-7]/type

DESCRIPTION
The Ethernet interface, /net/ethern, is a directory containing subdirectories, one for each distinct Ethernet packet type, and clone, addr, ifstats, and stats files. stats and ifstats are the same as in the subdirectories (see below). Reading addr returns the MAC address of this interface in hex with no punctuation and no trailing newline. The number n (optional in the bind) is the device number of the card, permitting multiple cards to be used on a single machine.

Each directory contains files to control the associated connection, receive and send data, and supply statistics. Incoming Ethernet packets are demultiplexed by packet type and passed up the corresponding open connection. Reading from the data file reads packets of that type arriving from the network. A read will terminate at packet boundaries. Each write to the data file causes a packet to be sent. The Ethernet address of the interface is inserted into the packet header as the source address.

A connection is assigned to a packet type by opening its ctl file and writing connect n where n is a decimal integer constant identifying the Ethernet packet type. A type of -1 enables the connection to receive copies of packets of all types. A type of -2 enables the connection to receive copies of the first 64 bytes of packets of all types. If multiple connections are assigned to a given packet type a copy of each packet is passed up each connection.

Some interfaces also accept unique options when written to the ctl (or clone) file; see the description of wavelan in plan9.ini(8).

Reading the ctl file returns the decimal index of the associated connection, 0 through 7. Reading the type file returns the decimal value of the assigned Ethernet packet type. Reading the stats file returns status information such as the Ethernet address of the card and general statistics, independent of the interface; ifstats contains device-specific data and statistics about the card.

An interface normally receives only those packets whose destination address is that of the interface or is the broadcast address, ff:ff:ff:ff:ff:ff. The interface can be made to receive all packets on the network by writing the string promiscuous to the ctl file. The interface remains promiscuous until the control file is closed. The extra packets are passed up connections only of types -1 and -2.

SOURCE
/sys/src/9/*/devether.c
NAME
flash - flash memory

SYNOPSIS
bind -a #F[n] /dev
/dev/flash
/dev/flash/part
/dev/flash/partctl

DESCRIPTION
The flash memory device serves a two-level directory, giving access to files representing part or all
of a bank of flash memory. A platform might have more than one bank of flash, numbered
starting from 0. The attach specifier n is a decimal integer that selects a particular bank of flash
(default: 0). Both NOR and NAND flash is supported. For both types of flash, the driver gives a
read/write/erase interface to the raw flash device, which can impose constraints on operations
beyond those imposed by the driver. Other drivers such as ftl(3) or logfs(3) implement any
higher-level format required, including ECC for NAND flash, for instance.

The top level directory contains a single directory named flash for bank 0, and flash\n for each
other bank n. It contains two files for each partition: a data file part and an associated control file
partctl, where part is the name of the partition. Each partition represents a region of flash
memory that starts and ends on a flash segment (erase unit) boundary. The system initially
creates a single standard partition flash representing the whole of flash memory, and the corre-
sponding control file flashctl. Other partitions can be created by writing to flashctl as
described below.

The data file part provides read and write access to the bytes on the system’s flash memory. Bytes
can be read and written on any byte boundary: the interface hides any alignment restrictions. A
read returns the value of the bytes at the current file offset, where zero is the start of the partition.
A write reprograms the flash to the given byte values, at the current file offset (relative to the start
of the partition), using the physical device’s reprogramming algorithm. An erased flash byte is
logically 0xFF (regardless of the conventions of the physical flash device). A write can change a
bit with value 1 to a 0, but cannot change a 0 bit to 1; that can only be done by erasing one or
more flash segments. NAND flash typically has restrictions on the number of writes allowed to a
page before requiring a block erase.

The control file partctl can be read and written. A read returns several lines containing decimal
and hexadecimal numbers (separated by white space) revealing the characteristics of memory
within the partition. The first line gives the the manufacturer ID, the flash device ID, the memory
width in bytes, and a string giving the flash type (currently either nor or nand). Subsequent lines
give characteristics of each group of erase units within the partition, where the erase units within a
group have the same properties. Each line gives the start and end (as byte addresses) of the erase
units in the region that lie within the partition, followed by the size in bytes of each erase unit,
which is followed for NAND flash by the size in bytes of a page. The sizes for NAND flash include
the extra bytes per page typically used to hold an ECC and block status. A write contains one of
the following textual commands:

add name start end
Create a new partition that ranges from start to end within the current partition.
Each value must be numeric (decimal, octal or hexadecimal) and a multiple of the erase unit size. Name must not be the name of an existing partition. On success, new files name and namectl will appear in the parent flash directory.

erase all
Erase the whole flash partition, setting all bytes to 0xFF, except those that are
hardware write-protected.

erase offset
Erase the segment that begins at the given offset within the partition, setting all
bytes to 0xFF, except those that are hardware write–protected. The offset is given
in bytes, but must be a multiple of the segment (erase unit) size.

protectboot [off]
By default the system prevents erase unit 0 of the flash from being erased or writ-
ten, assuming it contains the primary bootstrap. Writing this command with param-
eter off removes that protection. Writing protectboot with any other
parameter (or none) restores the protection. Note that a manufacturer might also have locked the flash in hardware, and that protection must be removed in a device-dependent way.

sync If the underlying device must buffer or cache (current devices do not), flush the buffer(s).

The syntax of all numbers is that of `strtol` (in `atof(2)`); the default base is 10.

SOURCE
/sys/src/*/devflash.c
/sys/src/*/flash*.c

SEE ALSO
flashfs(4), pafs(4)

DIAGNOSTICS
A write will return an error if an attempt is made to change a 0 bit to 1, or if the flash memory fails to be programmed correctly.

BUGS
The flash cannot be written if the kernel is executing directly from flash, because the physical flash cannot be read during programming, and the driver does not copy the programming code to DRAM.
NAME
floppy - floppy disk interface

SYNOPSIS
bind −a #f /dev

/dev/fd0disk
/dev/fd0ctl
/dev/fd1disk
/dev/fd1ctl
/dev/fd2disk
/dev/fd2ctl
/dev/fd3disk
/dev/fd3ctl

DESCRIPTION
The floppy disk interface serves a one-level directory giving access to up to four floppy disk drives. Each drive is represented by a data and control file. There are no partitions.

Messages accepted by the ctl file include:
eject Eject the floppy, if possible.
reset Reset the drive.
format type
       Format the floppy. The type sets the density and type of disk to be formatted; see format in prep(8).

A read of the ctl file returns a string describing the form factor of the disk, one of 3½DD, 3½HD, 5¼DD, or 5¼HD.

SOURCE
/sys/src/*/devfloppy.c
NAME
fs - file system devices

SYNOPSIS
bind -b #k /dev
/dev/fs
/dev/fs/ctl
/dev/fs/...
/dev/new

description
The fs driver builds complex disk files out of simpler disk files. Inspired by the Plan 9 file server kernel’s configuration strings, it provides device mirroring, partitioning, interleaving, and catenation for disk-based services like fossil(4) or venti(8).

The device is intended to be bound at /dev and initially contains a directory named fs, which in turn contains a cti file and one file per configured device.

Most control messages introduce a new device, here named new. The file arguments are interpreted in the name space of the writing process.

The device name new may be a single filename component (containing no slashes); in this case, the device is created under #k/fs. If new instead has the format dir/file, the device is made available at #k/dir/file. The directory dir goes away when the last device on it is removed with the del control message, but #k/fs will never be removed.

cat new files...
The device new corresponds to the catenation of files.

inter new files...
The device new corresponds to the block interleaving of files; an 8192-byte block size is assumed.

mirror new files...
The device new corresponds to a RAID-1-like mirroring of files. Writes to new are handled by sequentially writing the same data to the files from right to left (the reverse of the order in the control message). A failed write causes an eventual error return but does not prevent the rest of the writes to the other devices of the mirror set. Reads from new are handled by sequentially reading from the files from left to right until one succeeds. The length of the mirror device is the minimum of the lengths of the files.

part new file offset length

part new offset end
In the first form, the device new corresponds to the length units starting at offset in file. If offset+length reaches past the end of file, length is silently reduced to fit. Units are bytes. In the second form, a previous disk request must have defined the source file for further requests and the end of the device is determined by the end offset in the source file, and not by the device length. Units are as defined in the previous disk request. This form is accepted for compatibility with fdisk (in prep(8)) and sd(3) devices.

del old
Removes the device named old. The device will still be seen while in use. Further I/O attempts will fail with an error indication stating that the device is gone. When old is dir/*, all devices under dir are removed.

disk dir [ n file ]
makes dir implicit in new device names (i.e., it makes new mean dir/new by default). Optional argument n specifies the default unit (sector) size in bytes and the default source file for further partition devices. Default values are restored when the control file is closed.

clear
Discard all fs device definitions.

If the variable fsconfig is set in plan9.ini(8), fs will read its configuration from the file $fsconfig on the first attach. This is useful when the machine boots from a local file server that uses fs.
EXAMPLES
Use a previously partitioned disk, /dev/sdC0, making partition files available under
/dev/sdC0parts:

```
{  
edisk/sdC0parts 512 /dev/sdC0/data
disk/fdisk -p /dev/sdC0/data
  
  # now create plan 9 partitions
  
edisk/sdC0parts 512 /dev/sdC0parts/plan9
disk/prep -p /dev/sdC0parts/plan9

} > /dev/fs/ctl
```
Mirror the two disks /dev/sdC0/data and /dev/sdD0/data as /dev/fs/m0; similarly, mirror /dev/sdC1/data and /dev/sdD1/data as /dev/fs/m1:

```
  
edisk/mirror m0 /dev/sdC0/data /dev/sdD0/data > /dev/fs/ctl
edisk/mirror m1 /dev/sdC1/data /dev/sdD1/data > /dev/fs/ctl
```
Interleave the two mirrored disks to create /dev/fs/data:

```
  
edisk/inter data /dev/fs/m0 /dev/fs/m1 > /dev/fs/data
```
Run kfs(4) on the interleaved device:

```
  
disk/kfs -f /dev/fs/data
```
Save the configuration:

```
  
cp /dev/fs/ctl /dev/fd0disk
```
To load the configuration automatically at boot time, add this to plan9.ini:

```
  
  fsconfig=/dev/fd0disk
```

SEE ALSO
read in cat(1), dd(1), sd(3), fossil(4), fs(8), plan9.ini(8), prep(8), venti(8)

SOURCE
/sys/src/9/port/devfs.c

BUGS
Mirrors are RAID–like but not RAID. There is no fancy recovery mechanism and no automatic initial copying from a master drive to its mirror drives.

Each write system call on ctl may transmit at most one command.
NAME
i82365 – Personal Computer Memory Card Interface Association (PCMCIA) device

SYNOPSIS
bind -a #y /dev
/dev/pcm0attr
/dev/pcm0ctl
/dev/pcm0mem
/dev/pcm1attr
/dev/pcm1ctl
/dev/pcm1mem

DESCRIPTION
The i82365 driver provides an interface to an Intel 82365-compatible PCMCIA interface chip. This chip supports up to 2 PCMCIA slots, 0 and 1. Reading pcm[01]attr returns the contents of attribute memory. Reading or writing pcm[01]mem reads or writes RAM on the card. Reading pcm[01]ctl returns the card’s status.

This driver must be included to use PCMCIA devices such as the NE4100 Ethernet card. The individual card drivers make calls to routines in the PCMCIA driver.

SOURCE
/sys/src/9/pc/devi82365.c

SEE ALSO
plan9.ini(8)

BUGS
There is no driver for the Databook PCMCIA interface chip.
NAME
  ip, esp, gre, icmp, icmpv6, ipmux, rudp, tcp, udp – network protocols over IP

SYNOPSIS
  bind -a #Spec /net
  /net/ipifc
  /net/ipifc/clone
  /net/ipifc/stats
  /net/ipifc/n
  /net/ipifc/n/status
  /net/ipifc/n/ctl
  ... 
  /net/arp
  /net/bootp
  /net/iproute
  /net/ipselftab
  /net/log
  /net/nmdb
  /net/esp
  /net/gre
  /net/tcp
  ... 

DESCRIPTION
  The ip device provides the interface to Internet Protocol stacks. Spec is an integer from 0 to 15 identifying a stack. Each stack implements IPv4 and IPv6. Each stack is independent of all others: the only information transfer between them is via programs that mount multiple stacks. Normally a system uses only one stack. However multiple stacks can be used for debugging new IP networks or implementing firewalls or proxy services.

  All addresses used are 16-byte IPv6 addresses. IPv4 addresses are a subset of the IPv6 addresses and both standard ASCII formats are accepted. In binary representation, all v4 addresses start with the 12 bytes, in hex:

  00 00 00 00 00 00 00 00 00 00 00 ff ff

Configuring interfaces
  Each stack may have multiple interfaces and each interface may have multiple addresses. The /net/ipifc directory contains a clone file, a stats file, and numbered subdirectories for each physical interface.

  Opening the clone file reserves an interface. The file descriptor returned from the open(2) will point to the control file, ctl, of the newly allocated interface. Reading ctl returns a text string representing the number of the interface. Writing ctl alters aspects of the interface. The possible ctl messages are those described under Protocol directories below and these:

  bind ether path
    Treat the device mounted at path as an Ethernet medium carrying IP and ARP packets and associate it with this interface. The kernel will dial(2) path10x800 and path10x806 and use the two connections for IPv4 and ARP respectively.

  bind pkt
    Treat this interface as a packet interface. Assume a user program will read and write the data file to receive and transmit IP packets to the kernel. This is used by programs such as ppp(8) to mediate IP packet transfer between the kernel and a PPP encoded device.

  bind netdev path
    Treat this interface as a packet interface. The kernel will open path and read and write the resulting file descriptor to receive and transmit IP packets.

  bind loopback
    Treat this interface as a local loopback. Anything written to it will be looped back.

  unbind
    Disassociate the physical device from an IP interface.
add local mask remote mtu proxy
try local mask remote mtu proxy
Add a local IP address to the interface. Try adds the local address as a tentative address if it’s an IPv6 address. The mask, remote, mtu, and proxy arguments are all optional. The default mask is the class mask for the local address. The default remote address is local ANDed with mask. The default mtu (maximum transmission unit) is 1514 for Ethernet and 4096 for packet media. The mtu is the size in bytes of the largest packet that this interface can send. Proxy, if specified, means that this machine should answer ARP requests for the remote address. Ppp(8) does this to make remote machines appear to be connected to the local Ethernet.

remove local mask
Remove a local IP address from an interface.

mtu n
Set the maximum transfer unit for this device to n. The mtu is the maximum size of the packet including any medium-specific headers.

reassemble
Reassemble IP fragments before forwarding to this interface

iprouting n
Allow (n is missing or non-zero) or disallow (n is 0) forwarding packets between this interface and others.

bridge
Enable bridging (see bridge(3)).

promiscuous
Set the interface into promiscuous mode, which makes it accept all incoming packets, whether addressed to it or not.

connect type
marks the Ethernet packet type as being in use, if not already in use on this interface. A type of −1 means ‘all’ but appears to be a no-op.

addmulti Media−addr
Treat the multicast Media−addr on this interface as a local address.

remmulti Media−addr
Remove the multicast address Media−addr from this interface.

scanbs
Remove the wireless interface scan for base stations.

headersonly
Set the interface to pass only packet headers, not data too.

add6 v6addr pfx−len [onlink auto validlt preflt]
Add the local IPv6 address v6addr with prefix length pfx−len to this interface. See RFC 2461 §6.2.1 for more detail. The remaining arguments are optional:

    onlink flag: address is ‘on−link’
    auto flag: autonomous
    validlt valid life−time in seconds
    preflt preferred life−time in seconds

ra6 keyword value ...
Set IPv6 router advertisement (RA) parameter keyword’s value. Known keywords and the meanings of their values follow. See RFC 2461 §6.2.1 for more detail. Flags are true iff non−zero.

recvra flag: receive and process RAs.
sendra flag: generate and send RAs.
mflag flag: “Managed address configuration”, goes into RAs.
oflag flag: “Other stateful configuration”, goes into RAs.
maxraint “maximum time allowed between sending unsolicited multicast” RAs from the interface, in ms.
minraint “minimum time allowed between sending unsolicited multicast” RAs from the interface, in ms.
linkmtu “value to be placed in MTU options sent by the router.” Zero indicates none.
reachtime sets the Reachable Time field in RAs sent by the router. “Zero means unspecified (by this router).”
rxmitra sets the Retrans Timer field in RAs sent by the router. “Zero means unspecified (by this router).”

ttl default value of the Cur Hop Limit field in RAs sent by the router. Should be set to the “current diameter of the Internet.” “Zero means unspecified (by this router).”

routerlt sets the Router Lifetime field of RAs sent from the interface, in ms. Zero means the router is not to be used as a default router.

Reading the interface’s status file returns information about the interface, one line for each local address on that interface. The first line has 9 white-space-separated fields: device, mtu, local address, mask, remote or network address, packets in, packets out, input errors, output errors. Each subsequent line contains all but the device and mtu. See readipifc in ip(2).

Routing
The file iproute controls information about IP routing. When read, it returns one line per routing entry. Each line has six white-space-separated fields: target address, target mask, address of next hop, flags, tag, and interface number. The entry used for routing an IP packet is the one with the longest mask for which destination address ANDed with target mask equals the target address. The one-character flags are:

- 4 IPv4 route
- 6 IPv6 route
- i local interface
- b broadcast address
- u local unicast address
- m multicast route
- p point-to-point route

The tag is an arbitrary, up to 4 character, string. It is normally used to indicate what routing protocol originated the route.

Writing to /net/iproute changes the route table. The messages are:

flush Remove all routes.
tag string Associate the tag, string, with all subsequent routes added via this file descriptor.
add target mask nexthop Add the route to the table. If one already exists with the same target and mask, replace it.
remove target mask Remove a route with a matching target and mask.

Address resolution
The file /net/arp controls information about address resolution. The kernel automatically updates the v4 ARP and v6 Neighbour Discovery information for Ethernet interfaces. When read, the file returns one line per address containing the type of medium, the status of the entry (OK, WAIT), the IP address, and the medium address. Writing to /net/arp administers the ARP information. The control messages are:

flush Remove all entries.
add type IP=addr Media=addr Add an entry or replace an existing one for the same IP address.
del IP=addr Delete an individual entry.

ARP entries do not time out. The ARP table is a cache with an LRU replacement policy. The IP stack listens for all ARP requests and, if the requester is in the table, the entry is updated. Also, whenever a new address is configured onto an Ethernet, an ARP request is sent to help update the table on other systems.

Currently, the only medium type is ether.
Debugging and stack information
If any process is holding /net/log open, the IP stack queues debugging information to it. This is intended primarily for debugging the IP stack. The information provided is implementation-defined; see the source for details. Generally, what is returned is error messages about bad packets.

Writing to /net/log controls debugging. The control messages are:

set arglist  Arglist is a space-separated list of items for which to enable debugging. The possible items are: ppp, ip, fs, tcp, icmp, udp, compress, gre, tcpwin, tcpxrmt, udpmsg, ipmsg, and esp.
clear arglist  Arglist is a space-separated list of items for which to disable debugging.
only addr  If addr is non-zero, restrict debugging to only those packets whose source or destination is that address.

The file /net/ndb can be read or written by programs. It is normally used by ipconfig(8) to leave configuration information for other programs such as dns and cs (see ndb(8)). /net/ndb may contain up to 1024 bytes.

The file /net/ipselftab is a read-only file containing all the IP addresses considered local. Each line in the file contains three white-space-separated fields: IP address, usage count, and flags. The usage count is the number of interfaces to which the address applies. The flags are the same as for routing entries. Note that the ‘IPv4 route’ flag will never be set.

Protocol directories
The ip device supports IP as well as several protocols that run over it: TCP, UDP, RUDP, ICMP, GRE, and ESP. TCP and UDP provide the standard Internet protocols for reliable stream and unreliable datagram communication. RUDP is a locally-developed reliable datagram protocol based on UDP. ICMP is IP’s catch-all control protocol used to send low level error messages and to implement ping(8). GRE is a general encapsulation protocol. ESP is the encapsulation protocol for IPsec. It provided a reliable datagram service for communication between Plan 9 machines over IPv4, but is no longer part of the system.

Each protocol is a subdirectory of the IP stack. The top level directory of each protocol contains a clone file, a stats file, and subdirectories numbered from zero to the number of connections opened for this protocol.

Opening the clone file reserves a connection. The file descriptor returned from the open(2) will point to the control file, ctl, of the newly allocated connection. Reading ctl returns a text string representing the number of the connection. Connections may be used either to listen for incoming calls or to initiate calls to other machines.

A connection is controlled by writing text strings to the associated ctl file. After a connection has been established data may be read from and written to data. A connection can be actively established using the connect message (see also dial(2)). A connection can be established passively by first using an announce message (see dial(2)) to bind to a local port and then opening the listen file (see dial(2)) to receive incoming calls.

The following control messages are supported:

connect ip-address! port!r  local
   Establish a connection to the remote ip-address and port. If local is specified, it is used as the local port number. If local is not specified but !r is, the system will allocate a restricted port number (less than 1024) for the connection to allow communication with Unix login and exec services. Otherwise a free port number starting at 5000 is chosen. The connect fails if the combination of local and remote address/port pairs are already assigned to another port.

announce X  X is a decimal port number or *. Set the local port number to X and accept calls to X. If X is *, accept calls for any port that no process has explicitly announced. The local IP address cannot be set. Announce fails if the connection is already announced or connected.

bind X   X is a decimal port number or *. Set the local port number to X. This exists to support emulation of BSD sockets by the APE libraries (see pcc(1)) and is not
otherwise used.

ttl n  Set the time to live IP field in outgoing packets to n.
tos n  Set the service type IP field in outgoing packets to n.
ignoreadvice
        Don’t break (UDP) connections because of ICMP errors.
addmulti ifc–ip [ mcast–ip ]
        Treat ifc–ip on this multicast interface as a local address. If mcast–ip is present,
        use it as the interface’s multicast address.
remmulti ip  Remove the address ip from this multicast interface.

Port numbers must be in the range 1 to 32767.

Several files report the status of a connection. The remote and local files contain the IP
address and port number for the remote and local side of the connection. The status file con-
tains protocol–dependent information to help debug network connections. On receiving and error
or EOF reading or writing the data file, the err file contains the reason for error.

A process may accept incoming connections by open(2)ing the listen file. The open will block
until a new connection request arrives. Then open will return an open file descriptor which points
to the control file of the newly accepted connection. This procedure will accept all calls for the
given protocol. See dial(2).

TCP

TCP connections are reliable point–to–point byte streams; there are no message delimiters. A con-
nection is determined by the address and port numbers of the two ends. TCP ctl files support the
following additional messages:

hangup  close down this TCP connection
keepalive n  turn on keep alive messages. N, if given, is the milliseconds between keepalives
            (default 30000).
checksum n  emit TCP checksums of zero if n is zero; otherwise, and by default, TCP check-
            sums are computed and sent normally.
tcpporthogdefense onoff
            onoff of on enables the TCP port–hog defense for all TCP connections; onoff of
            off disables it. The defense is a solution to hijacked systems staking out ports
            as a form of denial–of–service attack. To avoid stateless TCP conversation hogs,
            ip picks a TCP sequence number at random for keepalives. If that number gets
            acked by the other end, ip shuts down the connection. Some firewalls, notably
            ones that perform stateful inspection, discard such out–of–specification keepalives, so
            connections through such firewalls will be killed after five minutes by the lack of keepalives.

UDP

UDP connections carry unreliable and unordered datagrams. A read from data will return the
next datagram, discarding anything that doesn’t fit in the read buffer. A write is sent as a single
datagram.

By default, a UDP connection is a point–to–point link. Either a connect establishes a local and
remote address/port pair or after an announce, each datagram coming from a different remote
address/port pair establishes a new incoming connection. However, many–to–one semantics is
also possible.

If, after an announce, the message headers is written to ctl, then all messages sent to the
announced port are received on the announced connection prefixed with the corresponding
structure, declared in <ip.h>:

        typedef struct Udphdr Udphdr;
        struct Udphdr
        {
            uchar      raddr[16];   /* V6 remote address and port */
            uchar      laddr[16];   /* V6 local address and port */
        }
Before a write, a user must prefix a similar structure to each message. The system overrides the user specified local port with the announced one. If the user specifies an address that isn’t a unicast address in /net/ipselftab, that too is overridden. Since the prefixed structure is the same in read and write, it is relatively easy to write a server that responds to client requests by just copying new data into the message body and then writing back the same buffer that was read.

In this case (writing headers to the ctl file), no listen nor accept is needed; otherwise, the usual sequence of announce, listen, accept must be executed before performing I/O on the corresponding data file.

**RUDP**

RUDP is a reliable datagram protocol based on UDP, currently only for IPv4. Packets are delivered in order. RUDP does not support listen. One must write either connect or announce followed immediately by headers to ctl.

Unlike TCP, the reboot of one end of a connection does not force a closing of the connection. Communications will resume when the rebooted machine resumes talking. Any unacknowledged packets queued before the reboot will be lost. A reboot can be detected by reading the err file. It will contain the message

```
  hangup address! port
```

where address and port are of the far side of the connection. Retransmitting a datagram more than 10 times is treated like a reboot: all queued messages are dropped, an error is queued to the err file, and the conversation resumes.

**RUDP** ctl files accept the following messages:

- **headers** Corresponds to the headers format of UDP.
- **hangup** IP port Drop the connection to address IP and port.
- **randdrop** [ percent ] Randomly drop percent of outgoing packets. Default is 10%.

**ICMP**

ICMP is a datagram protocol for IPv4 used to exchange control requests and their responses with other machines’ IP implementations. ICMP is primarily a kernel–to–kernel protocol, but it is possible to generate ‘echo request’ and read ‘echo reply’ packets from user programs.

**ICMPv6**

ICMPv6 is the IPv6 equivalent of ICMP. If, after an announce, the message headers is written to ctl, then before a write, a user must prefix each message with a corresponding structure, declared in <ip.h>:

```c
/*
 * user level icmpv6 with control message "headers"
 */
typedef struct Icmp6hdr Icmp6hdr;
struct Icmp6hdr {
  uchar laddr[IPaddrlen]; /* local address */
  uchar raddr[IPaddrlen]; /* remote address */
};
```

In this case (writing headers to the ctl file), no listen nor accept is needed; otherwise, the usual sequence of announce, listen, accept must be executed before performing I/O on the corresponding data file.

**GRE**

GRE is the encapsulation protocol used by PPTP. The kernel implements just enough of the protocol to multiplex it. Our implementation encapsulates in IPv4, per RFC 1702. Announce is not allowed in GRE, only connect. Since GRE has no port numbers, the port number in the connect is actually the 16 bit eproto field in the GRE header.
Reads and writes transfer a GRE datagram starting at the GRE header. On write, the kernel fills in the eproto field with the port number specified in the connect message.

ESP
ESP is the Encapsulating Security Payload (RFC 1827, obsoleted by RFC 4303) for IPsec (RFC 4301). We currently implement only tunnel mode, not transport mode. It is used to set up an encrypted tunnel between machines. Like GRE, ESP has no port numbers. Instead, the port number in the connect message is the SPI (Security Association Identifier (sic)). IP packets are written to and read from data. The kernel encrypts any packets written to data, appends a MAC, and prefixes an ESP header before sending to the other end of the tunnel. Received packets are checked against their MAC’s, decrypted, and queued for reading from data. In the following, secret is the hexadecimal encoding of a key, without a leading 0x. The control messages are:

esp alg secret Encrypt with the algorithm, alg, using secret as the key. Possible algorithms are: null, des_56_cbc, des3_cbc, and eventually aes_128_cbc, and aes_ctr.

ah alg secret Use the hash algorithm, alg, with secret as the key for generating the MAC. Possible algorithms are: null, hmac_sha1_96, hmac_md5_96, and eventually aes_xcbc_mac_96.

header Turn on header mode. Every buffer read from data starts with 4 unused bytes, and the first 4 bytes of every buffer written to data are ignored.

noheader Turn off header mode.

IP packet filter
The directory /net/ipmux looks like another protocol directory. It is a packet filter built on top of IP. Each numbered subdirectory represents a different filter. The connect messages written to the ctl file describe the filter. Packets matching the filter can be read on the data file. Packets written to the data file are routed to an interface and transmitted.

A filter is a semicolon-separated list of relations. Each relation describes a portion of a packet to match. The possible relations are:

proto=n the IP protocol number must be n.
data[n : m]=expr bytes n through m following the IP packet must match expr.

iph[n : m]=expr bytes n through m of the IP packet header must match expr.

ifc=expr the packet must have been received on an interface whose address matches expr.
src=expr The source address in the packet must match expr.
dst=expr The destination address in the packet must match expr.

Expr is of the form:

value
value | value | ...
value&mask
value | value&mask

If a mask is given, the relevant field is first ANDed with the mask. The result is compared against the value or list of values for a match. In the case of ifc, dst, and src the value is a dot- formatted IP address and the mask is a dot-formatted IP mask. In the case of data, iph and proto, both value and mask are strings of 2 hexadecimal digits representing 8-bit values.

A packet is delivered to only one filter. The filters are merged into a single comparison tree. If two filters match the same packet, the following rules apply in order (here ‘>’ means is preferred to):

1) protocol > data > source > destination > interface
2) lower data offsets > higher data offsets
3) longer matches > shorter matches
4) older > younger

So far this has just been used to implement a version of OSPF in Inferno and 6to4 tunnelling.

**Statistics**
The *stats* files are read only and contain statistics useful to network monitoring.

Reading `/net/ipifc/stats` returns a list of 19 tagged and newline-separated fields representing:

- forwarding status (0 and 2 mean forwarding off, 1 means on)
- output packets
- output packets discarded
- output packets with no route
- timed out fragments in reassembly queue
- requested reassemblies
- successful reassemblies
- failed reassemblies
- successful fragmentations
- unsuccessful fragmentations
- fragments created

Reading `/net/icmp/stats` returns a list of 26 tagged and newline-separated fields representing:

- messages received
- messages sent
- bad received messages
- transmission errors
- unreachables received
- unreachables sent
- time exceeded received
- time exceeded sent
- input parameter problems received
- input parameter problems sent
- source quenches received
- source quenches sent
- redirects received
- redirects sent
- echo requests received
- echo requests sent
- echo replies received
- echo replies sent
- timestamps received
- timestamps sent
- timestamp replies received
- timestamp replies sent
- address mask requests received
- address mask requests sent
- address mask replies received
- address mask replies sent

Reading `/net/tcp/stats` returns a list of 11 tagged and newline-separated fields representing:

- maximum number of connections
- segments sent
- total outgoing calls
- segments retransmitted
- total incoming calls
- retransmit timeouts
- number of established connections to be reset
- bad received segments
- number of currently established connections
- transmission failures
- segments received

Reading `/net/udp/stats` returns a list of 4 tagged and newline-separated fields representing:

- datagrams received
- malformed datagrams received
- datagrams received for bad ports
- datagrams sent

Reading `/net/ndb/stats` returns a list of 1 tagged number representing:

- header length errors

**SEE ALSO**

dial(2), ip(2), bridge(3), ndb(6), listen(8)
/lib/rfc/rfc2460 IPv6
/lib/rfc/rfc4291 IPv6 address architecture
/lib/rfc/rfc4443 ICMPv6

**SOURCE**

/sys/src/9/ip

**BUGS**

`ipmux` has not been heavily used and should be considered experimental. It may disappear in favor of a more traditional packet filter in the future.
NAME
  kbin - external keyboard input

SYNOPSIS
  bind -a #1 /dev
  /dev/kbin

DESCRIPTION
  The kbin device is a PC driver that serves a one-level directory containing a single file, kbin, which can be used to send keyboard scan codes to the kernel.

  Kbin is necessary for usb(4) drivers that handle keyboards. Keyboard input is processed as described in cons(3). The scan codes correspond to the PC keyboard used by the cons(3) driver and can be translated by the kbmap(3) device.

SEE ALSO
  cons(3), kbmap(3), keyboard(6)

SOURCE
  /sys/src/9/pc/devkbin.c
NAME
  kbmap - keyboard map

SYNOPSIS
  bind -a #x /dev

/dev/kbmap

DESCRIPTION
The *kbmap* device serves a one-level directory containing a single file, *kbmap*, representing the
kernel’s mapping of keyboard scan codes to Unicode characters (see *cons*(3) and *keyboard*(6)).
Reads return the current contents of the map. Each entry is one line containing three 11 character
numeric fields, each followed by a space: a table number, an index into the table (scan code), and
the decimal value of the corresponding Unicode character (0 if none). The table numbers are plat-
form dependent; they typically distinguish between unshifted and shifted keys. The scan code val-
ues are hardware dependent and can vary from keyboard to keyboard.
Writes to the file change the map. Lines written to the file must contain three space-separated
fields, representing the table number, scan code index, and Unicode character. Values are taken
to be decimal unless they start with 0x (hexadecimal) or 0 (octal). The Unicode character can also
be represented as ’x where x gives the UTF-8 representation of the character (see *utf*(6)), or as
^X to represent a control character.
The Unicode character can also be Mn to represent mouse button n. The map
/sys/lib/kbmap/mouse−fn maps the F1 through F5 keys to the three mouse buttons and
the two scroll wheel buttons. Similarly, mouse−csa maps the left Control, Start, and Alt keys to
the three mouse buttons. These maps are useful on laptops without three-button mice.

SEE ALSO
  *cons*(3), *keyboard*(6), *utf*(6)

FILES
  /sys/lib/kbmap/*

SOURCE
  /sys/src/9/port/devkbmap.c
NAME
kprof – kernel profiling

SYNOPSIS
bind -a #K /dev
/dev/kpctl
/dev/kpdata

DESCRIPTION
The kprof device provides simple profiling data for the operating system kernel. The data accumulates by recording the program counter of the kernel at each ‘tick’ of the system clock.

The file kpdata holds the accumulated counts as 4-byte integers in big-endian byte order. The size of the file depends on the size of kernel text. The first count holds the total number of clock ticks during profiling; the second the number of ticks that occurred while the kernel was running. The rest each hold the number of ticks the kernel program counter was within the corresponding 8-byte range of kernel text, starting from the base of kernel text.

The file kpctl controls profiling. Writing the string start to kpctl begins profiling; stop terminates it. The message startclr restarts profiling after zeroing the array of counts.

The program kprof (see prof(1)) formats the data for presentation.

EXAMPLE
The following rc(1) script runs a test program while profiling the kernel and reports the results.

bind -a '#K' /dev
echo start > /dev/kpctl
runtest
echo stop > /dev/kpctl
kprof /386/9pcdisk /dev/kpdata

SOURCE
/sys/src/9/port/devkprof.c

SEE ALSO
prof(1)
NAME
loopback - network link simulation

SYNOPSIS
bind -a #X /net
/net/loopbackn/[0-1]
/net/loopbackn/[0-1]/data
/net/loopbackn/[0-1]/ctl
/net/loopbackn/[0-1]/status
/net/loopbackn/[0-1]/stats

DESCRIPTION
The loopback interface, /net/loopbackn, is a directory containing two subdirectories, one for
each end of a simulated network link. The number n is the device number of the link, permitting
multiple links to be used on a single machine.

Each directory contains files to control the associated connection, receive and send data, monitor
the simulation parameters, and supply statistics.

The data files for the two directories are cross-connected. Writes to one are divided into packets
of at most a certain size, typically 32768 bytes, written to a flow-controlled output queue, trans-
ferred across the link, and put into an input queue where it is readable from the other data file.

Options are set by writing to the ctl file for the receiving end of the link, and are reported in the
same format by reading status. The following options are supported.

delay latency bytedelay
   Control the time a packet takes in the link. A packet n bytes long takes bytedelay * n
   nanoseconds to exit the output queue and is available for reading latency nanoseconds
   later.

droprate n
   Randomly drop approximately one out of n packets. If zero drop no packets.

indrop [01]
   Disallow or allow packets to be dropped if the input queue overflows.

limit n
   Set the input and output queues to hold at most n bytes.

reset
   Clear all of the statistics recorded for the link.

Reading stats returns a list of 4 tagged numbers representing:
   packets sent to this receiver
   bytes sent to this receiver
   packets dropped due to droprate
   packets dropped due to input queue overflows

SOURCE
/sys/src/9/port/devloopback.c
NAME
 lpt - parallel port interface for PC's

SYNOPSIS
 bind -a #L[123] /dev

 /dev/lpt[123]data
 /dev/lpt[123]dlr
 /dev/lpt[123]pcr
 /dev/lpt[123]psr

DESCRIPTION
 The lpt driver provides an interface to the parallel interface normally used for printers. The specifiers 1, 2, and 3 correspond to the parallel interfaces at PC ports 0x3bc, 0x378, and 0x278 respectively.

 Lpt?data is write only. Writing to it sends data to the interface. This file is sufficient for communicating with most printers.

 Lpt?dlr, lpt?pcr, and lpt?psr are used for fine control of the parallel port. Reading or writing these files corresponds to reading and writing the data latch register, printer control register, and printer status register. These are used by programs to drive special devices.

SOURCE
 /sys/src/9/pc/devlpt.c
NAME
mnt – attach to 9P servers

SYNOPSIS
#M

DESCRIPTION
The mount driver is used by the mount system call (but not bind; see bind(2)) to connect the
name space of a process to the service provided by a 9P server over a communications channel.
After the mount, system calls involving files in that portion of the name space will be converted by
the mount driver into the appropriate 9P messages to the server.

The mount system call issues session and attach(5) messages to the server to identify and validate
the user of the connection. Each distinct user of a connection must mount it separately; the mount
driver multiplexes the access of the various users and their processes to the service.

File-oriented system calls are converted by the kernel into messages in the 9P protocol. Within the
kernel, 9P is implemented by procedure calls to the various kernel device drivers. The mount
driver translates these procedure calls into remote procedure calls to be transmitted as messages
over the communication channel to the server. Each message is implemented by a write of the cor-
responding protocol message to the server channel followed by a read on the server channel to get
the reply. Errors in the reply message are turned into system call error returns.

A read(2) or write system call on a file served by the mount driver may be translated into more
than one message, since there is a maximum data size for a 9P message. The system call will
return when the specified number of bytes have been transferred or a short reply is returned.

The string #M is an illegal file name, so this device can only be accessed directly by the kernel.

SEE ALSO
bind(2)

SOURCE
/sys/src/9/port/devmnt.c

BUGS
When mounting a service through the mount driver, that is, when the channel being multiplexed is
itself a file being served by the mount driver, large messages may be broken in two.
NAME
mouse, cursor - kernel mouse interface

SYNOPSIS
bind -a #m /dev
/dev/mouse
/dev/mousein
/dev/mousectl
/dev/cursor

DESCRIPTION
The mouse device provides an interface to the mouse. There is also a cursor associated with the
screen; it is always displayed at the current mouse position.

Reading the mouse file returns the mouse status: its position and button state. The read blocks
until the state has changed since the last read. The read returns 49 bytes: the letter m followed by
four decimal strings, each 11 characters wide followed by a blank: x and y, coordinates of the
mouse position in the screen image; buttons, a bitmask with the 1, 2, and 4 bits set when the
mouse's left, middle, and right buttons, respectively, are down; and msec, a time stamp, in units
of milliseconds.

Writing the mouse file, in the same format, causes the mouse cursor to move to the position speci-
ified by the x and y coordinates of the message. The buttons and msec fields are ignored and may
be omitted.

Writes to the mousein file are processed as if they were generated by the mouse hardware itself,
as extra mouse events to be processed and passed back via the mouse file. The mousein file,
which is exclusive-use and may be opened only by the host owner, is intended for controlling
devices, such as USB mice, that are managed by user-level software. Each event should consist of
the letter m followed by delta x, delta y, and buttons as space-separated decimal numbers.

Writing to the mousectl file configures and controls the mouse. The messages are:

serial n   sets serial port n to be the mouse port.
ps2        sets the PS2 port to be the mouse port.
intellimouse uses the wheel on a Microsoft Intellimouse as the middle button.
ps2intellimouse is equivalent to a write of ps2 followed by a write of intellimouse.
accelerated [n] turns on mouse acceleration. N is an optional acceleration factor.
linear     turns off mouse acceleration.
res n      sets mouse resolution to a setting between 0 and 3 inclusive.
hwaccel on/off sets whether acceleration is done in hardware or software. By default,
        PS2 mice use hardware and serial mice use software. Some laptops
        (notably the IBM Thinkpad T23) don't implement hardware acceleration
        for external mice.
swap       swaps the left and right buttons on the mouse.
buttonmap xyz numbers the left, middle, and right mouse buttons x, y, and z, respectively.
            If xyz is omitted, the default map, 123, is used. Thus in the
default state writing buttonmap 321 swaps left and right buttons and
writing buttonmap 123 or just buttonmap restores their usual
meaning. Note that buttonmap messages are idempotent, unlike
swap.
reset      clears the mouse to its default state.

Not all mice interpret all messages; with some devices, some of the messages may be no-ops.

Cursor are described in graphics(2). When read or written from or to the cursor file, they are
represented in a 72-byte binary format. The first and second four bytes are little endian 32-bit
numbers specifying the x and y coordinates of the cursor offset; the next 32 bytes are the clr bit-
mask, and the last 32 bytes the set bitmask.

Reading from the cursor file returns the current cursor information. Writing to the cursor file
sets the current cursor information. A write of fewer than 72 bytes sets the cursor to the default,
an arrow.
The mouse and cursor files are multiplexed by rio(1) to give the illusion of a private mouse to each of its clients. The semantics are otherwise the same except that notification of a window resize is passed to the application using a mouse message beginning with r rather than m; see rio(4) for details.

To cope with pointing devices with only two buttons, when the shift key is pressed, the right mouse button generates middle-button events.

SOURCE
/sys/src/9/port/devmouse.c

SEE ALSO
rio(4)

BUGS
The cursor format is big endian while the rest of the graphics interface is little endian.
NAME
  pipe - two-way interprocess communication

SYNOPSIS
  bind # | dir

  dir/data
  dir/data1

DESCRIPTION
  An attach(5) of this device allocates two new cross-connected I/O streams, dir/data and dir/data1.
  Data written to one channel becomes available for reading at the other. Write boundaries are preserved: each read terminates when the read buffer is full or after reading the last byte of a write, whichever comes first.
  Writes are atomic up to a certain size, typically 32768 bytes, that is, each write will be delivered in a single read by the recipient, provided the receiving buffer is large enough.
  If there are multiple writers, each write is guaranteed to be available in a contiguous piece at the other end of the pipe. If there are multiple readers, each read will return data from only one write.
  The pipe(2) system call performs an attach of this device and returns file descriptors to the new pipe's data and data1 files. The files are open with mode ORDWR.

SEE ALSO
  pipe(2)

SOURCE
  /sys/src/9/port/devpipe.c
NAME
pnp - Plug 'n' Play ISA and PCI Interfaces

SYNOPSIS
bind -a '#$' /dev
/dev/pci/bus.dev.fnctl
/dev/pci/bus.dev.fnraw
/dev/pnp/ctl
/dev/pnp/csnctl
/dev/pnp/csnraw
...

DESCRIPTION
This device provides a limited interface to the PCI bus and Plug 'n' Play ISA devices.

PCI Interface
PCI devices are addressed logically by a bus number, a device number on that bus, and a function number within the device. The set of all such device functions may be enumerated by traversing the /dev/pci directory; the driver serves two files for each function. These are a control file (/dev/pci/bus.dev.fnctl) which may be read for a textual summary of the device function, and a 'raw' file (/dev/pci/bus.dev.fnraw) which may be used to read or write the raw contents of PCI configuration space.

The first field of a PCI control file contains the class, sub-class and programming interface values for the device function, expressed as 2-digit hexadecimal values, and separated by periods. The second field yields the vendor ID and device ID, each as 4-digit hex numbers, separated by a slash. The third field is the associated interrupt line in decimal. The remainder of the line enumerates any valid base address registers for the function, using two fields for each. In the first field, the index of the register is followed by a colon, and then the value of the register itself. The following field gives the associated size of the memory (or I/O space) that is mapped by the register.

Plug 'n' Play
Plug 'n' Play ISA devices are discovered by sending a fixed 'unlock' sequence over an I/O port, and then reading back data from another port. An arbitration algorithm is used to separate out the individual cards and enumerate them in turn. Each card is assigned a unique number, called a CSN, in the range 1–255 as a result of enumeration. Cards also have a fixed 64 bit identification number, set by the manufacturer, which is used by the arbitration algorithm to resolve conflicts. The first 32 bits describe the type of the card, and the second 32 bits form a serial number for the particular instance of that card type. When formatted textually, it appears as 3 upper-case letters (typically representing the manufacturer), followed by 4 hex digits, then a period, then 8 hex digits. The substring before the period is the card type, and the substring after the period is the serial number.

The enumeration algorithm needs to be enabled by specifying the port number to write the unlock sequence out on. This can be configured to take place at boot time by adding a line like the following to plan9.ini:

```
 pnp0=port=0x203
```

Here port should be chosen to not conflict with any existing devices. It must be in the range 0x203–0x3ff. Alternatively, one can use the following command:

```
echo port 0x203 > /dev/pnp/ctl
```

Note that a side-effect of PnP enumeration is to reset the configuration state of all such cards; any settings made by a Plug and Play BIOS will be lost. Reading the file /dev/pnp/ctl returns one of the strings enabled port or disabled.

For each enumerated card, two files are served in /dev/pnp. A control file (/dev/pnp/csnctl) may be read to determine the ID of the card, and a raw file (/dev/pnp/csnraw) may be read to obtain the configuration data associated with the card. It is intended that the control file should take commands which set the various configurable resources of the card, but this has not been implemented yet.
A mechanism is provided for configuring cards via `plan9.ini(8)`. A line of the form `pnpn=idstring` ... will cause the driver to look for the card named by `idstring` and, if found, assign it the CSN `n`. The intention is that any additional text after the `idstring` is interpreted as if it was written to the card’s `ctl` file, but this is not yet implemented.

**EXAMPLES**

To list all PCI functions:
```
cat /dev/pci/*ctl
```

To find just the PCI video card (class 3):
```
grep '^03' /dev/pci/*ctl
```

**SOURCE**
```
/sys/src/9/port/devnpn.c
```

**SEE ALSO**
`pci(8)`

**BUGS**

Access to the I/O and memory regions of a PCI device is not provided.

The ability to set a Plug ‘n’ Play card’s configurable settings has not been implemented.

There should be a user program for identifying and configuring Plug ‘n’ Play cards.
NAME
proc – running processes

SYNOPSIS
bind #p /proc
    /proc/trace
    /proc/n/args
    /proc/n/ctl
    /proc/n/fd
    /proc/n/fpregs
    /proc/n/kregs
    /proc/n/mem
    /proc/n/note
    /proc/n/noteid
    /proc/n/notepg
    /proc/n/ns
    /proc/n/proc
    /proc/n/profile
    /proc/n/regs
    /proc/n/segment
    /proc/n/status
    /proc/n/text
    /proc/n/wait
...

DESCRIPTION
The proc device serves a two-level directory structure. The first level contains the trace file (see below) and numbered directories corresponding to pids of live processes; each such directory contains a set of files representing the corresponding process.

The mem file contains the current memory image of the process. A read or write at offset o, which must be a valid virtual address, accesses bytes from address o up to the end of the memory segment containing o. Kernel virtual memory, including the kernel stack for the process and saved user registers (whose addresses are machine-dependent), can be accessed through mem. Writes are permitted only while the process is in the Stopped state and only to user addresses or registers.

The read-only proc file contains the kernel per-process structure. Its main use is to recover the kernel stack and program counter for kernel debugging.

The files regs, fpregs, and kregs hold representations of the user-level registers, floating-point registers, and kernel registers in machine-dependent form. The kregs file is read-only.

The read-only fd file lists the open file descriptors of the process. The first line of the file is its current directory; subsequent lines list, one per line, the open files, giving the decimal file descriptor number; whether the file is open for read (r), write, (w), or both (rw); the type, device number, and qid of the file; its I/O unit (the amount of data that may be transferred on the file as a contiguous piece; see iounit(2)), its I/O offset; and its name at the time it was opened.

The read-only ns file contains a textual representation of the process’s file name space, in the format of namespace(6) accepted by newns (see auth(2)). The last line of the file identifies the current working directory of the process, in the form of a cd command (see rc(1)). The information in this file is based on the names files had when the name space was assembled, so the names it contains may be inaccessible if the files have been subsequently renamed or rearranged.

The read-only segment file contains a textual display of the memory segments attached to the process. Each line has multiple fields: the type of segment (Stack, Text, Data, Bss, etc.); one-letter flags such as R for read-only, if any; starting virtual address, in hexadecimal; ending virtual address, and reference count.

The read-only status file contains a string with twelve fields, each followed by a space. The fields are:
- the process name and user name, each 27 characters left justified
- the process state, 11 characters left justified (see ps(1))
- the six 11-character numbers also held in the process’s #/c/cpu time file
- the amount of memory used by the process, except its stack, in units of 1024 bytes
- the base and current scheduling priority, each 11 character numbers

The read-only args file contains the arguments of the program when it was created by exec(2). If the program was not created by exec, such as by fork(2), its args file will be empty. The format of the file is a list of quoted strings suitable for tokenize; see getfields(2).

The text file is a pseudonym for the file from which the process was executed; its main use is to recover the symbol table of the process.

The wait file may be read to recover records from the exiting children of the process in the format of await (see wait(2)). If the process has no extant children, living or exited, a read of wait will block. It is an error for a process to attempt to read its own wait file when it has no children. When a process’s wait file is being read, the process will draw an error if it attempts an await system call; similarly, if a process is in an await system call, its wait file cannot be read by any process.

The read-only profile file contains the instruction frequency count information used for multiprocess profiling; see tprof in prof(1). The information is gleaned by sampling the program’s user-level program counter at interrupt time.

Strings written to the note file will be posted as a note to the process (see notify(2)). The note should be less than ERRLEN – 1 characters long; the last character is reserved for a terminating NUL character. A read of at least ERRLEN characters will retrieve the oldest note posted to the process and prevent its delivery to the process. The notepg file is similar, but the note will be delivered to all the processes in the target process’s note group (see fork(2)). However, if the process doing the write is in the group, it will not receive the note. The notepg file is write-only.

The textual noteid file may be read to recover an integer identifying the note group of the process (see RFNOTEG in fork(2)). The file may be written to cause the process to change to another note group, provided the group exists and is owned by the same user.

The file /proc/trace can be opened once and read to see trace events from processes that have had the string trace written to their ctl file. Each event produces, in native machine format, the pid, a type, and a time stamp (see /sys/include/trace.h and /sys/src/cmd/trace.c).

Control messages
Textual messages written to the ctl file control the execution of the process. Some require that the process is in a particular state and return an error if it is not.

stop Suspend execution of the process, putting it in the Stopped state.
start Resume execution of a Stopped process.

waitstop Do not affect the process directly but, like all other messages ending with stop, block the process writing the ctl file until the target process is in the Stopped state or exits. Also like other stop control messages, if the target process would receive a note while the message is pending, it is instead stopped and the debugging process is resumed.

startstop Allow a Stopped process to resume, and then do a waitstop action.

hang Set a bit in the process so that, when it completes an exec(2) system call, it will enter the Stopped state before returning to user mode. This bit is inherited across fork(2) and exec(2).

close n Close file descriptor n in the process.
closefiles Close all open file descriptors in the process.
nohang Clear the hang bit.
noswap Don’t allow this process to be swapped out. This should be used carefully and sparingly or the system could run out of memory. It is meant for processes that can’t be swapped, like the ones implementing the swap device and for processes containing sensitive data.
kill Kill the process the next time it crosses the user/kernel boundary.
private Make it impossible to read the process’s user memory. This property is inherited on fork, cleared on exec(2), and is not otherwise resettable.
pri \( n \) Set the base priority for the process to the integer \( n \).
wired \( n \) Wire the process to processor \( n \).
trace Without an argument, toggle trace event generation for this process into /proc/trace (see below). With a zero argument, tracing for the proc is turned off, with a non-zero numeric argument, it is turned on.
period \( nu \) Set the real-time scheduling period of the process to \( nu \), where \( n \) is an optionally signed number containing an optional decimal point and \( u \) is one of s, ms, us, µs, ns, or empty. The time is interpreted, respectively, as seconds, milliseconds, microseconds, nanoseconds, or, in the case of an absent units specifier, as nanoseconds. If the time specifier is signed, it is interpreted as an increment or decrement from a previously set value. See also the admit command below.
deadline \( nu \) Set the real-time deadline interval of the process to \( nu \), where \( n \) and \( u \) are interpreted as for period above.
cost \( nu \) Set the real-time cost (maximum CPU time per period) of the process to \( nu \), where \( n \) and \( u \) are interpreted as for period above.
sporadic Use sporadic scheduling for the real-time process. The description of the admit command below contains further details.
yieldonblock Make the real-time process yield on blocking I/O.
    The description of the admit command below contains further details.
admit Given real-time period, deadline and cost are set (an unset deadline will set deadline to period), perform a schedulability test and start scheduling the process as a real-time process if the test succeeds. If the test fails, the write will fail with error set to the reason for failure.
event Add a user event to the /proc/trace file.

**Real-time scheduling**

Real-time processes are periodically released, giving them a higher priority than non-real-time processes until they either give up the processor voluntarily, they exhaust their CPU allocation, or they reach their deadline. The moment of release is dictated by the period and whether the process is sporadic or not. Non-sporadic processes are called periodic and they are released precisely at intervals of their period (but periods can be skipped if the process blocks on I/O). Sporadic processes are released whenever they become runnable (after being blocked by sleep() or I/O), but always at least an interval of period after the previous release.

The deadline of a real-time process specifies that the process must complete within the first deadline seconds of its period. The deadline must be less than or equal to the period. If it is not specified, it is set to the period.

The cost of a real-time process describes the maximum CPU time the process may use per period. A real-time process can give up the CPU before its deadline is reached or its allocation is exhausted. It does this by calling sleep(0). If yieldonblock is specified, it also does it by executing
any blocking system call. Yieldonblock is assumed for sporadic processes.

Of the released processes, the one with the earliest deadline has the highest priority. Care should be taken using spin locks (see lock(2)) because a real-time process spinning on a lock will not give up the processor until its CPU allocation is exhausted; this is unlikely to be the desired behavior.

When a real-time process reaches its deadline or exhausts its CPU allocation, it remains schedu-

able, but at a very low priority.

The priority is interpreted by Plan 9’s multilevel process scheduler. Priorities run from 0 to 19, with higher numbers representing higher priorities. A process has a base priority and a running priority which is less than or equal to the base priority. As a process uses up more of its allocated time, its priority is lowered. Unless explicitly set, user processes have base priority 10, kernel pro-

cesses 13. Children inherit the parent’s base priority.

FILES
/sys/src/*mem.h
/sys/src/*dat.h
/sys/include/trace.h

SEE ALSO
trace(1), debugger(2), mach(2), cons(3)

SOURCE
/sys/src/port/devproc.c
NAME
  root - the root file system

SYNOPSIS
  /
  /boot
  /dev
  /env
  /net
  /net.alt
  /proc
  /root
  /srv

DESCRIPTION
  The syntax # is illegal, so this device can only be accessed directly by the kernel.

  This device is set up by the kernel to be the root of the name space. The names in the one-level
  tree are mostly just place-holders, to allow a place to bind(2) to. The exception is /boot, which
  contains /boot/boot and any files /boot/boot might need. The kernel does an exec(2) of
  /boot/boot when initializing.

SOURCE
  /sys/src/9/port/devroot.c
NAME
    rtc - real-time clock and non-volatile RAM

SYNOPSIS
    bind #r /dev
    /dev/rtc
    /dev/nvram

DESCRIPTION
    The rtc device supports devices with real-time clocks and non-volatile RAM.
    The rtc file behaves just like /dev/time (see cons(3)). The real-time clock is maintained on-
    board; /dev/time is set from the file server. Neither is necessarily more accurate.
    The nvram file provides (if permission allows) access to the local non-volatile RAM. For example,
    boot(8) reads the machine's key from there (see auth(8)).

SEE ALSO
    auth(8), boot(8)

SOURCE
    /sys/src/9/*/devrtc.c
NAME
sd - storage device interface

SYNOPSIS
bind #S /dev
/dev/sdctl
/dev/sdCu/ctl
/dev/sdCu/raw
/dev/sdCu/data
...

DESCRIPTION
The storage device interface serves a two-level directory giving access to multiple storage units, typically ATA(PI) or SCSI discs. Each unit is accessed via files in the directory named by the controller to which it is attached, C, and by its unit number u. The controller naming convention for ATA(PI) units starts with the first controller being named C, the second D, etc. up to a maximum of 4 controllers ([C−F]); legacy controllers are always ‘C’ and ‘D’. There can be a maximum of 2 units per ATA(PI) controller ([01]). The controller naming convention for SCSI units starts with the first controller being named 0, the second 1, etc. up to a maximum of 16 controllers ([0−9a−f]). There can be a maximum of 16 units per SCSI controller ([0−9a−f]).

Units are not accessed before the first attach. Units may be individually attached using the attach specifier, for example

bind -a '#SsdD0' /dev

An attach without a specifier will cause the driver to scan for all possible units before processing the rest of the name.

The subdirectory for each unit contains two files, ctl and raw. In addition, if the unit is a direct-access disc of some type it may be split into partitions and the subdirectory may contain a file per partition. By default, the partition data will exist for such media.

Partitions are added and deleted by writing to the ctl file

    part name start-sector end-sector
delpart name

The default data partition may be deleted. A partition cannot be deleted if a process has it open. If a change of removable media is detected, the new media cannot be opened until all open partitions on the old media are closed.

Partitions are usually created using fdisk and prep(8); the convention is to name non-Plan 9 partitions after their corresponding operating systems (e.g., /dev/sdC0/dos) and Plan 9 partitions according to their function (e.g., /dev/sdC0/swap). The example in prep(8) shows how this is done.

Reading the ctl file returns at least one line of textual information about the unit. The first line will always be prefixed by inquiry and will give a manufacturer and model number if possible. A line prefixed by config will be returned for appropriate media, e.g. for ATA(PI) units the remainder of the line contains configuration information from the device's identify command (config and capabilities) and also the available I/O transfer options; this is a diagnostic aid. A line prefixed by geometry will be returned for appropriate media; at least two numbers will follow, the first being the number of sectors contained in the unit and the second the sector size in bytes. Any remaining information on the geometry line is unit-dependent, for instance, head, cylinder and sector counts for ATA discs. If any partitions are defined for the media, their name, start-sector and end-sector will be returned, prefixed by part.

    % cat /dev/sdD0/ctl
    inquiry KENWOOD CD-ROM UCR-421 208E10/20/99 7.39 2 M0
    config 85C0 capabilities 0F00 dma 00550004 dmactl 00000000
    geometry 242725 2352
    part data 0 242725
    %
The use of DMA and multi-sector read/write commands may be enabled and disabled on ATA(PI) units by writing to the ctl file dma and rwm respectively followed by on or off. For example, to enable DMA on a unit that supports it:

% echo 'dma on' >/dev/sd00/ctl

If supported by the unit, the standby timer may be enabled:

% echo 'standby T' >/dev/sdC0/ctl

where T is the standby timer period in seconds. T must be between 30 and 1200, or can be 0 to disable the timer.

The raw file is used to execute an arbitrary command on the unit at a low level. This is used by programs such as scuzz(8) to manipulate devices that do not fit the simple storage model or for maintenance purposes. The following steps may be taken to execute a command:

- Write the command to the raw file;
- Read or write data associated with the command, according to the direction of the transfer.
- Read the raw file to retrieve the status of the command, returned as a text integer.

Reading /dev/sdctl yields information about each controller, one line per controller. Writing 'config message' to /dev/sdctl passes message to the legacy configuration machinery, used to set attributes such as IRQ, port and size. Writing 'ctltype message' to /dev/sdctl passes message to ctltype's wtopctl function with a nil sdev argument, where ctltype is a known controller type such as ata or scsi. Writing 'sctlletter message' to /dev/sdctl passes message to sctlletter's wtopctl function with an sdev argument corresponding to the named controller, where ctlletter is a known controller letter such as C or 0.

SOURCE
/sys/src/*/sd*.c
/sys/src//*/sd*.hc

SEE ALSO
scuzz(8)

BUGS
LUNs (logical unit numbers) are not implemented. For (S)ATA drives, LUNs are not merely ignored but are actively prevented from working except for INQUIRY commands.

The 4 controller limit for ATA(PI) is not enforced.

No account is taken of some buggy ATA PCI controllers such as the CMD640.

ATA(PI) units come up with DMA and multi-sector read/write capability disabled.
NAME
sdahci - AHCI (Advanced Host Controller Interface) SATA (Serial ATA) storage device drivers

SYNOPSIS
bind -a #S /dev
/dev/sdctl
/dev/sdEn/ctl
/dev/sdEn/raw
/dev/sdEn/data
...

DESCRIPTION
The sdahci driver provides access to AHCI devices via the sd(3) interface. The AHCI programming
interface supports up to 32 hot-swappable ATAPI or hard disk–like devices per controller. The
legacy IDE interface provided by sdata.c supports up to four drives which are not hot-
swappable. Controller drive letters are assigned from E onward.

AHCI controllers are detected automatically. Currently Intel and AMD controllers are detected.
Intel controllers need to have AHCI enabled in the BIOS. For ich parts this typically means
enabling enhanced mode and AHCI. For ESB (Enterprise South Bridge)–based parts, only enhanced
mode needs to be enabled. Intel ich9–based AHCI does not support hot swapping and drives
must be connected to the lowest–numbered free port.

The top level control file, /dev/sdctl, supports the following control messages for sdahci:
iahci debug       Toggle debug messages. Default is off.
iahci idprint     Toggle printing of drive identification messages. Default is on. Prints short
                   messages when a drive is identified or removed.
iahci aprint      Print verbose ATAPI debugging messages. Default is off.

The device–level ctrl file supports:
flushcache        Send the ATA/ATAPI FLUSH CACHE command (0xe7 or 0xea). This command
                   may take up to 60 seconds to complete.
identify          Send the ATA/ATAPI IDENTIFY DEVICE command (0xec). If device infor-
                   mation has changed, the new size, features and serial will be noted. If
                   changed, I/O on existing file descriptors will result in the error string media
                   or partition has changed.
mode speed         Change the connection speed to one of auto, satai or sataii.
nop                Send the ATA NOP command (0) if the device supports it. Per standard, the
                   result is always an error.
smart              Send the ATA/ATAPI SMART RETURN STATUS command (0xda). This will
                   fail unless SMART is enabled on the drive.
smartdisable       Disable SMART on the drive. SMART is a persistent property of the drive.
smartenable        Enable SMART on the drive.
state state        Force a transition to the named state. The states are:
                   null                   ignored (may only be reached manually);
                   missing                not detected;
                   new                    powered down or newly discovered;
                   ready                  ready for commands;
                   reset                  being reset gently;
                   portreset              being fully reset;
                   offline                device failed portreset (a port reset will be attempted
                                           periodically).

For devices present at boot, the transition is from state new to state ready.

SOURCE
/sys/src/9/pc/sdiahci.c

SEE ALSO
sd(3), 9load(8)
BUGS

None of enclosure management, LED control and port multipliers are supported.
ATAPI devices may not be reset when they have outstanding commands.
NAME
sdaoe - ATA-over-Ethernet (AoE) storage device interface

SYNOPSIS
bind -a #S /dev
echo config switch on spec l type aoe//dev/aoe/shelf.slot >/dev/sdctl
echo config switch off spec l >/dev/sdctl
/dev/sd/0/ctl
/dev/sd/0/raw
/dev/sd/0/data
...
addaoe letter unit

DESCRIPTION
Sdaoe has a few quirks because network-attached storage can't be enumerated as directly-
attached storage can. The default first controller letter for AoE devices is e. Each sdaoe device
must be configured explicitly.
Addaoe packages up the switch on invocation as an rc script.
To boot from an AoE root, the sd device must be configured on boot by either PXE booting or
booting from directly-attached storage and adding two configuration lines to plan9.ini(8) for
aoeif, listing the names of the Ethernet interface(s) to use, and aoedev=letter!#æ/aoe/lun.

EXAMPLES
To configure target (LUN) 42.0 on #S/sde0,
    echo config switch on spec e type aoe//dev/aoe/42.0 >/dev/sdctl
To turn this device off,
    echo config switch off spec e >/dev/sdctl
To boot using target 42.0 as #S/sde0 and as root, over Ethernet interfaces 0 and 1,
    aoeif=ether0 ether1
    aoedev=e!#æ/aoe/42.0

SOURCE
/sys/src/9/port/sdaoec.c

SEE ALSO
aoe(3), sd(3), 9load(8), snoopy(8)

BUGS
It is not currently possible to boot from an AoE target without an external bootstrap like PXE.
NAME
sdp - secure datagram protocol

SYNOPSIS
bind -a #Espec /net
/net/sdp/clone
/net/sdp/log
/net/sdp/n
/net/sdp/n/data
/net/sdp/n/control
/net/sdp/n/ctl
/net/sdp/n/rstats
/net/sdp/n/stats
/net/sdp/n/status
...

DESCRIPTION
The sdp device provides the interface to the Secure Datagram Protocol (SDP). SDP (un)compresses and (de-)encrypts packets. Spec is an integer from 0 to 15 identifying a stack. Each stack is independent of all others; the only information transfer between them is via programs that mount multiple stacks. Normally a system uses only one stack. However multiple stacks can be used for debugging new networks or implementing firewalls or proxy services.

The top level directory contains a clone file, a log file, and subdirectories numbered from zero to the number of connections opened for this protocol.

Opening the clone file reserves a connection. The file descriptor returned from the open(2) will point to the control file, ctl, of the newly allocated connection. Reading ctl returns a text string representing the number of the connection. Connections may be used either to listen for incoming calls or to initiate calls to other machines.

A connection is controlled by writing text strings to the associated ctl file. After a connection has been established data may be read from and written to data. A connection can be actively established using the connect message (see also dial(2)). A connection can be established passively by first using an announce message (see dial(2)) to bind to a local port and then opening the listen file (see dial(2)) to receive incoming calls.

The following control messages are supported:

accept file Accept an incoming encrypted connection on file, typically a data file.
dial file Initiate a new encrypted connection on file, typically a UDP data file.
drop permil Randomly drop approximately one of every permil output packets, thus simulating network errors.
cipher alg Use alg as ciphering algorithm for this connection. Choices are null, des_56_cbc, rc4_128, and rc4_256.
auth alg Use alg as authentication algorithm for this connection. Choices are null, hmac_sha1_96, and hmac_md5_96.
comp alg Use alg as compression algorithm for this connection. Choices are null and thwack.
insecret secret Use secret as the secret for decrypting incoming packets.
outsecret secret Use secret as the secret for encrypting outgoing packets.

SEE ALSO
dial(2), ip(3)

SOURCE
/sys/src/9/port/devsdp.c
NAME
segment - long lived memory segments

SYNOPSIS
bind 'g' /mnt/segment

#g/seg1
#g/seg1/ctl
#g/seg1/data
#g/seg2
#g/seg2/ctl
#g/seg2/data
...

DESCRIPTION
The segment device provides a 2-level file system representing long-lived sharable segments that
processes may segattach(2). The name of the directory is the class argument to segattach.

New segments are created under the top level using create (see open(2)). The DMDIR bit must be set in the permissions. Remove(2)'ing the directory makes the segment no longer available for segattach. However, the segment will continue to exist until all processes using it either exit or segdetach it.

Within each segment directory are two files, data and ctl. Reading and writing data affects the contents of the segment. Reading and writing ctl retrieves and sets the segment's properties.

There is only one control message, which sets the segment's virtual address and length in bytes:

va address length
Address is automatically rounded down to a page boundary and length is rounded up to end the
segment at a page boundary. The segment will reside at the same virtual address in all processes
sharing it. When the segment is attached using segattach, the address and length arguments are
ignored in the call; they are defined only by the va control message. Once the address and length
are set, they cannot be reset.

Reading the control file returns a message of the same format with the segment's actual start
address and length.

Opening data or reading ctl before setting the virtual address yields the error "segment not yet
allocated".

The permissions check when segattach is equivalent to the one performed when opening data
with mode ORDWR.

EXAMPLE
Create a one megabyte segment at address 0x10000000:
% bind 'g' /mnt/segment
% mkdir /mnt/segment/example
% echo 'va 0x10000000 0x10000000' > /mnt/segment/example/ctl

Put the string “hi mom” at the start of the segment:
% echo -n hi mom > /mnt/segment/example/data

Attach the segment to a process:
{
    ulong va;

    va = segattach(0, "example", 0, 0);

}

SEE ALSO
segattach(2)

SOURCE
/sys/src/9/port/devsegment.c
NAME
  srv – server registry

SYNOPSIS
  bind #s /srv

  #s/service1
  #s/service2
  ...

DESCRIPTION
  The srv device provides a one-level directory holding already-open channels to services. In effect, srv is a bulletin board on which processes may post open file descriptors to make them available to other processes.

  To install a channel, create a new file such as /srv/mysrv and then write a text string (suitable for strtoul; see atof(2)) giving the file descriptor number of an open file. Any process may then open /srv/mysrv to acquire another reference to the open file that was registered.

  An entry in srv holds a reference to the associated file even if no process has the file open. Removing the file from /srv releases that reference.

  It is an error to write more than one number into a server file, or to create a file with a name that is already being used.

EXAMPLE
  To drop one end of a pipe into /srv, that is, to create a named pipe:

  ```c
  int fd, p[2];
  char buf[32];
  
  pipe(p);
  fd = create("/srv/namedpipe", OWRITE, 0666);
  fprintf(fd, "%d", p[0]);
  close(fd);
  close(p[0]);
  fprintf(p[1], "hello");
  ```

  At this point, any process may open and read /srv/namedpipe to receive the hello string. Data written to /srv/namedpipe can be received by executing

  ```c
  read(p[1], buf, sizeof buf);
  ```

  in the above process.

SOURCE
  /sys/src/9/port/devsrv.c
NAME
ssl - SSL record layer

SYNOPSIS
bind -a #D /net
/net/ssl/clone
/net/ssl/n
/net/ssl/n/ctl
/net/ssl/n/data
/net/ssl/n/encalgs
/net/ssl/n/hashalgs
/net/ssl/n/secretin
/net/ssl/n/secretout

DESCRIPTION
The SSL device provides the interface to the Secure Socket Layer device implementing the record
layer protocol of SSLv2 (but not the handshake protocol, which is responsible for mutual authen-
tication and key exchange.) The ssl device can be thought of as a filter providing optional encryp-
tion and anti-tampering.

The top level directory contains a clone file and subdirectories numbered from zero to the num-
er of connections configured. Opening the clone file reserves a connection. The file descriptor
returned from the open(2) will point to the control file, ctl, of the newly allocated connection. Reading the ctl file returns a text string representing the number of the connection.

A connection is controlled by writing text strings to the associated ctl file. After a connection
has been established data may be read from and written to the data file.

The SSL protocol provides a stream connection that preserves read/write boundaries. As long
as reads always specify buffers that are of equal or greater lengths than the writes at the other end
of the connection, one write will correspond to one read.

Options are set by writing control messages to the ctl file of the connection.

The following control messages are supported:

fd open-file-descriptor
 Run the SSL protocol over the existing file descriptor.

alg cryptoalgs
 Connections start in alg clear which means no encryption or digesting. Writing alg
sha to the control file turns on SHA-1 digest authentication for the data channel. Similarly, writing alg rc4_128 enables encryption. Both can be turned on at once by alg
sha rc4_128. The digest mode sha may be replaced by md5. The encryption mode
rc4_128 may be replaced by rc4_40, rc4_128, rc4_256, des_40_ecb, des_40_cbc, des_56_ecb, and des_56_cbc. The mode may be changed at any
time during the connection.

secretin base64-secret
 The secret for decrypting and authenticating incoming messages can be specified either as
a base64 encoded string by writing to the control file, or as a binary byte string using the
interface below.

secretout base64-secret
 The secret for encrypting and hashing outgoing messages can be specified either as a
base64 encoded string by writing to the control file, or as a binary byte string using the
interface below.

Before enabling digesting or encryption, shared secrets must be agreed upon with the remote side,
one for each direction of transmission, and loaded as shown above or by writing to the files
secretin and secretout. If either the incoming or outgoing secret is not specified, the other secret is
assumed to work for both directions.

The encryption and hash algorithms actually included in the kernel may be smaller than the set pre-
 sented here. Reading encalgs and hashalgs will give the actual space-separated list of algorithms
implemented.

SEE ALSO

listen(8), dial(2)

SOURCE

/sys/src/9/port/devssl.c

BUGS

Messages longer than 4096 bytes are truncated.
NAME
tls - TLS1 and SSL3 record layer

SYNOPSIS
bind -a #a /net
/net/tls/clone
/net/tls/encalgs
/net/tls/hashalgs
/net/tls/n
/net/tls/n/ctl
/net/tls/n/data
/net/tls/n/hand
/net/tls/n/stats
/net/tls/n/status

DESCRIPTION
The TLS device implements the record layer protocols of Transport Layer Security version 1.0 and Secure Sockets Layer version 3.0. It does not implement the handshake protocols, which are responsible for mutual authentication and key exchange. The tls device can be thought of as filters providing optional encryption and anti-tampering.

The top level directory contains a clone file and subdirectories numbered from zero through at least the last active filter. Opening the clone file reserves a filter. The file descriptor returned from the open(2) will point to the control file, ctl, of the newly allocated filter. Reading the ctl file returns a text string containing the number of the filter directory.

The filter initially cannot be used to pass messages and will not encrypt or digest messages. It is configured and controlled by writing commands to ctl.

The following commands are supported:

cfd open-fd vers
Pass record messages over the communications channel open-fd. Initially, outgoing messages use version vers format records, but incoming messages of either version are accepted. Valid versions are 0x300 for SSLv3.0 and 0x301 for TLSv1.0 (which could be known as SSLv3.01.) This command must be issued before any other command and before reading or writing any messages; it may only be executed once.

version vers
Use vers format records for all future records, both outgoing and incoming. This command may only be executed once.

secret hashalg encalg isclient secretdata
Set up the digesting and encryption algorithms and secrets. Hashalg and encalg must be algorithm names returned by the corresponding files. Secretdata is the base-64 encoded (see encode(2)) secret data used for the algorithms. It must contain at least enough data to populate the secrets for digesting and encrypting. These secrets are divided into three categories: digest secrets, keys, and initialization vectors. The secrets are packed in this order, with no extra padding. Within each category, the secret for data traveling from the client to the server comes first. The incoming and outgoing secrets are automatically selected by devtls based on the isclient argument, which must be non-zero for the client of the TLS handshake, and zero for the server.
This command must be issued after version, and may be issued more than once. At least one new secret command must be issued before each changecipher command; similarly, at least one new secret command must precede each incoming changecipher message.

changepassword
Enable outgoing encryption and digesting as configured by the previous secret command. This command sends a changecipher message.

opened
Enable data messages. This command may be issued any number of times, although only
the first is significant. It must follow at least one successful changeCipher command.

```
alert alertno
   Send an alert message. Alertno may be a valid alert code for either SSLv3.0 or TLSv1.0, and is mapped to an appropriate code for the protocol in use. If it is a fatal alert, the filter is set into an error state.
```

Application messages and handshake messages are communicated using data and hand, respectively. Only one open(2) of hand is allowed at a time.

Any record layer headers and trailers are inserted and stripped automatically, and are not visible from the outside. The device tries to synchronize record boundaries with reads and writes. Each read will return data from exactly one record, and will return all of the data from the record as long as the buffer is big enough. Each write will be converted into an integral number of records, with all but potentially the last being maximal size. The maximum record length supported is 16384 bytes. This behavior is not specified in the protocols, and may not be followed by other implementations.

If a fatal alert message is received, or a fatal alert command issued, the filter is set into an error state. All further correspondence is halted, although some pending operations may not be terminated. Operations on data will fail with a 'tls error', and operations on hand will fail with a textual decoding of the alert. The current non-fatal alert messages are 'close notify', 'no renegotiation', and 'handshake canceled by user'. Receipt of one of these alerts cause the next read on hand to terminate with an error. If the alert is 'close notify', all future reads will terminate with a tls hungup error. A 'close notify' alert command will terminate all future writes or reads from data with a 'tls hungup' error.

If an error is encountered while reading or writing the underlying communications channel, the error is returned to the offending operation. If the error is not 'interrupted', the filter is set into the error state. In this case, all future operations on hand will fail with a 'channel error'.

When all file descriptors for a filter have been closed, the session is terminated and the filter reclaimed for future use. A 'close notify' alert will be sent on the underlying communications channel unless one has already been sent or the filter is in the error state.

Reading stats or status returns information about the filter. Each datum is returned on a single line of the form tag: data. Stats returns the number of bytes communicated by the data and hand channels. The four lines returned are tagged by, in order, DataIn, DataOut, HandIn, and HandOut. Status returns lines following tags: State, Version, EncIn, HashIn, NewEncIn, NewHashIn, EncOut, HashOut, NewEncOut, and NewHashOut. State's value is a string describing the status of the connection, and is one of the following: 'Handshaking', 'Established', 'RemoteClosed', 'LocalClosed', 'Alerting', 'Errored', or 'Closed'. Version's give the hexadecimal record layer version in use. The Enc and Hash fields return name of the current algorithms in use or ready to be used, if any.

Reading encalgs and hashalgs will give the space-separated list of algorithms implemented. This will always include clear, meaning no encryption or digesting. Currently implemented encryption algorithms are 'rc4_128' and '3des_ede_cbc'. Currently implemented hashing algorithms are 'md5' and 'sha1'.

SEE ALSO
   listen(8), dial(2), pushtls(2)

SOURCE
   /sys/src/9/port/devtls.c
NAME
twsi – two-wire serial interface (TWSI) and inter-integrated circuit (I²C) interface

SYNOPSIS
bind -a #² /dev
dev/twsi*

DESCRIPTION
The twsi device serves a one-level directory containing one file per TWSI or I²C bus. Bytes written are transmitted on the bus; bytes received from the bus are queued and delivered by reading. Seeking to a given offset before reading or writing causes the twsi device to use that offset as a TWSI slave address for a subsequent read(2) or write call.

FILES
#²/twsi*

SOURCE
/sys/src/*/devtwsi.c

BUGS
10-bit addressed devices are not supported.
No slave mode.
Setting the bus rate is not supported.
NAME
  uart, eia - serial communication control

SYNOPSIS
  bind -a #t /dev
  /dev/eia0
  /dev/eia0ctl
  /dev/eia0status
  /dev/eial
  /dev/eialctl
  /dev/eialstatus
  ...

DESCRIPTION
  The serial line devices serve a one-level directory, giving access to the serial ports. Device n is accessed through eian (the data file), eiactl (the control file), and eianstatus (the read-only status file). Reads of the data file will block until at least one byte is available. The control file configures the port. It accepts the following commands:

  b n  Set the baud rate to n.
  c n  Set hangup on DCD if n is non-zero; else clear it.
  d n  Set DTR if n is non-zero; else clear it.
  e n  Set hangup on DSR if n is non-zero; else clear it.
  f    Flush output queue.
  h    Close input and output queues.
  i n  Enable/disable the FIFOs. If n is zero the FIFOs are disabled; otherwise n is taken as a trigger level for the FIFOs. The trigger levels supported are device dependant, but usually include 1, 4 and 8. An unrecognised, but non-zero, value of n causes the maximum-supported trigger level to be set.
  k n  Send a break lasting n milliseconds.
  l n  Set number of bits per byte to n. Legal values are 5, 6, 7, or 8.
  m n  Obey modem CTS signal if n is non-zero; else clear it.
  n    Make writes non-blocking.
  p c  Set parity to odd if c is o, to even if c is e; else set no parity.
  q n  Set input and output queue limits to n.
  r n  Set RTS if n is non-zero; else clear it.
  s n  Set number of stop bits to n. Legal values are 1 or 2.
  w n  Set the uart clock timer to n times 100us.

  The status files contain a textual representation of the status of the line, in the format of the commands used on the control file.

SOURCE
  /sys/src/9/port/devuart.c
  /sys/src/9/*/uart*.c
NAME
usb - USB Host Controller Interface

SYNOPSIS
bind -a #u /dev
/dev/usb
/dev/usb/ctl
/dev/usb/epN.M
/dev/usb/epN.M/data
/dev/usb/epN.M/ctl
...

DESCRIPTION
The Universal Serial Bus is a complex yet popular bus for connecting all kind of devices to a computer. It is a four-wire tree-shaped bus that provides both communication and (limited) power to devices. Branching points in the tree are provided by devices called hubs. Hubs provide ports where USB devices (also hubs) can be attached.

Most PCs have one or more USB controllers called host controllers. Each one has a built-in hub called a root hub providing several ports. In some cases, more hubs are built-in and attached to a root hub port. The topology of the network is a tree with at most 127 nodes, counting both internal and leaf nodes.

Host controllers come in four flavours: UHCI and OHCI for USB 1 (up to 12 Mb/s), EHCI for USB 2 (up to 480 Mb/s) and XHCI for USB 3 (up to 5 Gb/s). We currently support all but XHCI, which is still quite new.

The USB bus is fully controlled by the host; all devices are polled. Hubs are passive in the sense that they do not poll the devices attached to them. The host polls those devices and the hubs merely route the messages.

Devices may be added to or removed from the bus at any time. When a device is attached, the host queries it to determine its type and speed. The querying process is standardized. The first level of querying is the same for all devices, the next is somewhat specialized for particular classes of devices (such as mice, keyboards, or audio devices). Specialization continues as subclasses and subsubclasses are explored.

Enumeration of the bus and initial configuration of devices is done by a user level program, usbd(4). Device drivers are implemented by separate user programs, although some of them may be statically linked into usbd.

The kernel device described in this page is responsible for providing I/O for using the devices through so called endpoints. Access to the host controller is hidden from user programs, which see just a set of endpoints. After system initialization, some endpoints are created by the device to permit I/O to root hubs. All other devices must be configured by usbd.

Devices and Endpoints
A device includes one or more functions (e.g., audio output, volume control buttons, mouse input, etc.) Communication with device functions is performed by some combination of issuing control requests to, sending data to, and receiving data from device endpoints. Endpoints can be understood as addresses in the bus. There are several types:

Control Their main use is to configure devices. Writing a message with a specific format (specified in the USB specification) issues a request to the device. If the request implies a reply, a read can be made next to retrieve the requested data (if the write succeeded).

Interrupt Used to send and receive messages to or from a specific device function (e.g., to read events from a mouse).

Bulk Used to send and receive larger amounts of data through streams (e.g., to write blocks to a disk).

Isochronous Used to send and receive data in a timely manner (e.g., to write audio samples to a speaker).
All USB devices include at least a control endpoint to perform device configuration. This is called the setup endpoint or endpoint zero. After configuring a device, other endpoints may be created as dictated by the device to perform actual I/O.

**Operation**

Bus enumeration and device configuration is performed by `usbd(4)` and not by this driver. The driver provides an interface to access existing endpoints (initially those for the built-in root hubs), to create and configure other ones, and to perform I/O through them.

Each directory `/dev/usb/epN.M` represents an endpoint, where N is a number identifying a device and M is a number identifying one of its endpoints.

For each device attached to the bus, and configured by `usbd(4)`, an endpoint zero (a setup endpoint) is provided at `/dev/usb/epN.0` for configuring the device. This is always a control endpoint and represents the device itself.

The device driver may use the setup endpoint to issue control requests and perhaps to create more endpoints for the device. Each new endpoint created has its own directory as said above. For example, if the driver for the device `/dev/usb/epN.0` creates the endpoint number 3 for that device, a directory `/dev/usb/epN.3` will be available to access that endpoint.

All endpoint directories contain two files: `data` and `ctl`. The former has mode bit DMEXCL set and can be open by only one process at a time.

**data**

The `data` file is used to perform actual I/O. In general, reading from it retrieves data from the endpoint and writing into it sends data to the endpoint. For control endpoints, writing to this file issues a control request (which may include `data`); if the request retrieves data from the device, a following read on the file will provide such data.

USB errors reported by the endpoint upon I/O failures are passed to the user process through the error string. I/O stalls not resulting from an error, usually an indication from the device, are reported by indicating that the number of bytes transferred has been zero. In most cases, the correct course of action after noticing the stall is for the device driver to issue a ‘clear halt’ request (see `uninstall in usbd(2)`) to resume I/O. The most common error is `crc`/timeout indicating problems in communication with the device (e.g., a physical detach of the device or a wiring problem).

For control and isochronous transfers, there is an implicit timeout performed by the kernel and it is not necessary for applications to place their own timers. For other transfer types, the kernel will not time out any operation by default (but see the `timeout control request`).

**ctl and status**

The `ctl` file can be read to learn about the endpoint. It contains information that can be used to locate a particular device (or endpoint). It also accepts writes with textual control requests described later.

This may result from the read of an endpoint control file:

```
(enabled control rw speed full maxpkt 64 pollival 0
 samplesz 0 hz 0 hub 1 port 3 busy
 storage csp 0x500608 vid 0x951 did 0x1613 Kingston 'DT 101 II'
)
```

The first line contains status information. The rest is information supplied by `usbd(4)` as an aid to locate devices. The status information includes:

**Device state**

One of `config`, `enabled`, and `detached`. An endpoint starts in the `config` state, and accepts control commands written to its `ctl` file to configure the endpoint. When configured, the state is `enabled` and the `data` file is used as described above (several control requests can still be issued to its `ctl` file, but most will not be accepted from now on). Upon severe errors, perhaps a physical detachment from the bus, the endpoint enters the `detached` state and no further I/O is accepted on it. Files for an endpoint (including its directory) vanish when the device is detached and its files are no longer open. Root hubs may not be detached.
Endpoint type: control, iso, interrupt, or bulk, indicating the type of transfer supported by the endpoint.

Endpoint mode: One of r, w, or rw, depending on the direction of the endpoint (in, out, or inout).

Speed: low (1.5 Mb/s), full (12 Mb/s), or high (480 Mb/s).

Maximum packet size: Used when performing I/O on the data file.

Polling interval: The polling period expressed as a number of µframes (for high-speed endpoints) or frames (for low- and full-speed endpoints). Note that a µframe takes 125 µs while a frame takes 1 ms. This is only of relevance for interrupt and isochronous endpoints. This value determines how often I/O happens. Note that the control request adjusting the polling interval does not use these units, to make things easier for USB device drivers.

Sample size: Number of bytes per I/O sample (isochronous endpoints only).

Frequency: Number of samples per second (Hertz).

Hub address: Device address of the hub where the device is attached.

Port number: Port number (in the hub) where the device is attached.

Usage: busy while the data file is open and idle otherwise. This is useful to avoid disturbing endpoints already run by a device driver.

The second line contains information describing the device:

Class name: As provided by the device itself.

CSP: Class, Subclass, and Protocol for the device. If the device contains different functions and has more CSPs, all of them will be listed. The first one is that of the device itself. For example, a mouse and keyboard combo may identify itself as a keyboard but then include two CSPs, one for the keyboard and another one for the mouse.

Vendor and device identifiers: Provided by the device and identifying the manufacturer and type of device.

For example, to find a mouse not yet in use by a driver, scan the ctl files for enabled, idle, and csp 0x020103. A mouse belongs to class 3 (in the least significant byte), human interface device, subclass 1, boot, protocol 2, mouse (protocol 1 would be the keyboard). USB class, subclass and proto codes can be found at http://www.usb.org.

Control requests:

Endpoint control files accept the following requests. In most cases the driver does not issue them, leaving the task to either usbd(4) or the usb driver library documented in usb(2).

detach: Prevent further I/O on the device (delete the endpoint) and remove its file interface as soon as no process is using their files.

maxpkt n: Set the maximum packet size to n bytes.

pollival n: Only for interrupt and isochronous endpoints. Set the polling interval as a function of the value n given by the endpoint descriptor. The interval value used is the period n in bus time units for low- and full-speed interrupt endpoints. Otherwise, the actual interval is $2 sup n$ and not n. Bus time units are 1 ms for low- and full-speed endpoints and 125 µs for high-speed endpoints. In most cases, the device driver may ignore all this and issue the control request supplying the polling interval value as found in the endpoint descriptor. The kernel adjusts the value according to the endpoint configuration and converts it into the number of frames or µframes between two consecutive polls.

samplesz n: Use n as the number of bytes per sample.

clearsz n: Use n as the number of samples per second.

ntds n: Use n as the number of transactions per frame (or µframe), as reported by the descriptor.

clrhalt: Clear the halt condition for an endpoint. Used to recover from a stall caused by a device to signal its driver (usually due to an unknown request or a failure to
info string Replaces description information in ctl with string. Usbd(4) uses this to add device descriptions.

address Tell this driver that the device has been given an address, which causes the device to enter the enabled state.

name str Generates an additional file name, str, for the data file of the endpoint. This file name appears in the root directory of the /dev tree. For example, this is used by the audio device driver to make the data file also available as /dev/audio.

debug n Enable debugging of the endpoint. N is an integer; larger values make diagnostics more verbose. 0 stops debugging diagnostics. 1 causes just problem reports. Bigger values report almost everything.

timeout n Enable time-outs for the endpoint. Transfers are timed out by the kernel after n ms. This should not be used for control and isochronous endpoints, which are always timed out.

Setup endpoints (those represented by epN.0 directories) also accept the following requests:

new n type mode Creates a new endpoint with number n of the given type (ctl, bulk, intr, or iso). Mode may be r, w, or rw, which creates, respectively, an input, output, or input/output endpoint.

speed {low|full|high} Set the endpoint speed to full, low, or high, respectively.

hub Tell this driver that the endpoint corresponds to a hub device.

Setup endpoints for hub devices also accept his request:

newdev {low|full|high} port Create a new setup endpoint to represent a new device. The first argument is the device speed. Port is the port number where the device is attached (the hub is implied by the endpoint where the control request is issued).

The file /dev/usb/ctl provides all the information provided by the various ctl files when read. It accepts several requests that refer to the entire driver and not to particular endpoints:

debug n Sets the global debug flag to n.

dump Dumps data structures for inspection.

FILES
#u/usb root of the USB interface

SOURCE
/sys/src/9/pc/usb.h
/sys/src/9/pc/devusb.c
/sys/src/9/pc/usb?hci.c

SEE ALSO
usb(2), usb(4), usbd(4), plan9.init(8)

BUGS
Isochronous input streams are not implemented for OHCI.
Some EHCI controllers drop completion interrupts and so must be polled, which hurts throughput.
Not heavily exercised yet.
NAME
gva - VGA controller device

SYNOPSIS
bind #v /dev
/dev/vgabios
/dev/vgactl
/dev/vgaovl
/dev/vgaovlctl

DESCRIPTION
The VGA device allows configuration of a graphics controller on a PC. Vgactl allows control over higher-level settings such as display height, width, depth, controller and hardware-cursor type. Along with the I/O-port registers provided by arch(3), it is used to implement configuration and setup of VGA controller cards. This is usually performed by vga(8).

Vgabios provides read-only access to the low 640kB of memory, where the VGA and other BIOS ROMs are located.

Writing strings to vgactl configures the VGA device. The following are valid commands.
size XxYxZ chan
Set the size of the screen image to be X pixels wide and Y pixels high. Each pixel is Z bits as specified by chan, whose format is described in image(6).

actualsize XxY
Set the physical size of the display to be X pixels wide by Y pixels high. This message is optional; it is used to implement panning and to accommodate displays that require the in-memory screen image to have certain alignment properties. For example, a 1400x1050 screen with a 1408x1050 in-memory image will use size 1408x1050 but actualsize 1400x1050.

panning mode
Depending on whether mode is on or off, enable or disable panning in a virtual screen. If panning is on and the screen's size is larger than its actualsize, the displayed portion of the screen will pan to follow the mouse. Setting the panning mode after the first attach of the #i driver has no effect.

type ctrl
Set the type of VGA controller being used. Ctrl is one of ark200pv, clgd542x, clgd546x, ct65545, cyber938x, hiqvideo, mach64xx, mga2164, neomagic, nvidia, s3, and t2r4.

Note that this list does not indicate the full set of VGA chips supported. For example, s3 includes the 86C801/5, 86C928, Vision864, and Vision964. It is the job of vga(8) to recognize which particular chip is being used and to initialize it appropriately.

hwgc gc
Set the type of hardware graphics cursor being used. Gc is one of ark200pvhwgc, bt485hwgc, clgd542xhwgc, clgd546xhwgc, ct65545hwgc, cyber938xhwgc, hiqvideohwgc, mach64xxhwgc, mga2164hwgc, neomagicwhwc, nvidiahwgc, rgb524hwgc, s3hwgc, t2r4hwgc, tvp3020hwgc, and tvp3026hwgc. A value of off disables the cursor. There is no software cursor.

palettedepth d
Set the number of bits of precision used by the VGA palette to d, which must be either 6 or 8.

blank
Blank the screen. This consists of setting the hardware color map to all black as well as, on some controllers, setting the VGA hsync and vsync signals so as to turn off VESA DPMS-compliant monitors. The screen also blanks after 30 minutes of inactivity. The screen can be unblanked by moving the mouse.
blanktime minutes
   Set the timeout before the screen blanks; the default is 30 minutes. If minutes is zero, blanking is disabled.

hwaccel mode
   Depending on whether mode is on or off, enable or disable whether hardware acceleration (currently for rectangle filling and moving) used by the graphics engine. The default setting is on.

hwblank mode
   Depending on whether mode is on or off, enable or disable the use of DPMS blanking (see blank above).

linear size align
   Use a linear screen aperture of size size aligned on an align-byte boundary.

drawinit
   Initialize the graphics hardware. This must be sent after setting the type.

Reading vgactl returns the current settings, one per line.

Some VGA cards support overlay graphics. Writing strings to vgaovlctl configures such cards. The following are valid overlay control commands:

openctl
   opens the overlay device.

configure w h format
   allocates resources inside the driver to support an overlay area of width w and height h pixels. Currently, the only supported format is YUYV packed. In YUYV two pixels are encoded by their separate Y values and their combined U and V values. The size of the two pixels is 32 bits.

enable x y w h
   enables drawing data on the display through the overlay mode. The data is drawn at position x, y and has a width and height of w, h respectively.

closectl
   terminates overlay control.

Overlay data can be written to vgaovl.

EXAMPLES
   The following disables hardware acceleration.
      echo hwaccel off > /dev/vgactl

SOURCE
   /sys/src/9/pc/devvga.c

SEE ALSO
   arch(3), vga(8)

BUGS
   The hardware graphics cursor on the et4000 does not work in 2x8-bit mode.
NAME
   intro - introduction to file servers

DESCRIPTION
   A Plan 9 file server provides a file tree to processes. This section of the manual describes servers
   than can be mounted in a name space to give a file-like interface to interesting services. A file
   server may be a provider of a conventional file system, with files maintained on permanent stor-
   age, or it may also be a process that synthesizes files in some manner.

SEE ALSO
   bind(1)
NAME
acme - control files for text windows

SYNOPSIS
acme [-ab][ -c ncol][ -E varfont][ -F fixfont][ -l file | file ...]

DESCRIPTION
The text window system acme(1) serves a variety of files for reading, writing, and controlling
windows. Some of them are virtual versions of system files for dealing with the virtual console; others
control operations of acme itself. When a command is run under acme, a directory holding these
files is mounted on /mnt/acme (also bound to /mnt/wsys) and also /dev; the files mentioned
here appear in both those directories.

Some of these files supply virtual versions of services available from the underlying environment,
in particular the character terminal files cons(3). (Unlike in rio(1), each command under acme sees
the same set of files; there is not a distinct /dev/cons for each window.) Other files are unique
to acme.

acme is a subdirectory used by win (see acme(1)) as a mount point for the acme files associated
with the window in which win is running. It has no specific function under acme itself.

cons is the standard and diagnostic output file for all commands run under acme. (Input for
commands is redirected to /dev/null.) Text written to cons appears in a window
labeled dir/ Errors, where dir is the directory in which the command was run. The win-
don is created if necessary, but not until text is actually written.

consctl
Is an empty unwritable file present only for compatibility; there is no way to turn off 'echo',
for example, under acme.

index
holds a sequence of lines of text, one per window. Each line has 5 decimal numbers, each
formatted in 11 characters plus a blank—the window ID; number of characters (runes) in
the tag; number of characters in the body; a 1 if the window is a directory, 0 otherwise; and
a 1 if the window is modified, 0 otherwise—followed by the tag up to a newline if present.
Thus at character position 5x12 starts the name of the window. If a file has multiple
zeroxed windows open, only the most recently used will appear in the index file.

label
is an empty file, writable without effect, present only for compatibility with rio.

new A directory analogous to the numbered directories (q.v.). Accessing any file in new creates
a new window. Thus to cause text to appear in a new window, write it to
/dev/new/body. For more control, open /dev/new/ctl and use the interface
described below.

Each acme window has associated a directory numbered by its ID. Window IDs are chosen sequen-
tially and may be discovered by the ID command, by reading the ctl file, or indirectly through
the index file. The files in the numbered directories are as follows.

addr may be written with any textual address (line number, regular expression, etc.), in the for-
mat understood by button 3 but without the initial colon, including compound addresses,
to set the address for text accessed through the data file. When read, it returns the value
of the address that would next be read or written through the data file, formatted as 2
decimal numbers m and n, each formatted in 11 characters plus a blank. M and n are the
character (not byte) offsets of the beginning and end of the address, which would be
expressed in acme 's input language as #m, #n. Thus a regular expression may be evalu-
ated by writing it to addr and reading it back. The addr address has no effect on the
user's selection of text.

body holds contents of the window body. It may be read at any byte offset. Text written to
body is always appended; the file offset is ignored.

c1t may be read to recover the five numbers as held in the index file, described above, plus
three more fields: the width of the window in pixels, the name of the font used in the win-
dow, and the width of a tab character in pixels. Text messages may be written to ctl to
affect the window. Each message is terminated by a newline and multiple messages may be sent in a single write.

`addr=dot` Set the addr address to that of the user’s selected text in the window.
`clean` Mark the window clean as though it has just been written.
`dirty` Mark the window dirty, the opposite of clean.
`cleartag` Remove all text in the tag after the vertical bar.
`del` Equivalent to the Del interactive command.
`delete` Equivalent to the Delete interactive command.
`dot=addr` Set the user’s selected text in the window to the text addressed by the addr address.

`dump command` Set the command string to recreate the window from a dump file.
`dumpdir directory` Set the directory in which to run the command to recreate the window from a dump file.
`get` Equivalent to the Get interactive command with no arguments; accepts no arguments.
`limit=addr` When the CTL file is first opened, regular expression context searches in addr addresses examine the whole file; this message restricts subsequent searches to the current addr address.
`mark` Cancel nomark, returning the window to the usual state wherein each modification to the body must be undone individually.
`menu` Maintain Undo, Redo, and Put in the left half of the tag. (This is the default for file windows.)
`name name` Set the name of the window to name.
`nomark` Turn off automatic ‘marking’ of changes, so a set of related changes may be undone in a single Undo interactive command.
`nomenu` Do not maintain Undo, Redo, and Put in the left half of the tag. (This is the default for directory and error windows.)
`noscroll` Turn off automatic ‘scrolling’ of the window to show text written to the body.
`put` Equivalent to the Put interactive command with no arguments; accepts no arguments.
`scroll` Cancel a noscroll message, returning the window to the default state wherein each write to the body file causes the window to ‘scroll’ to display the new text.
`show` Guarantee at least some of the selected text is visible on the display.

`data` is used in conjunction with addr for random access to the contents of the body. The file offset is ignored when writing the data file; instead the location of the data to be read or written is determined by the state of the addr file. Text, which must contain only whole characters (no ‘partial runes’), written to data replaces the characters addressed by the addr file and sets the address to the null string at the end of the written text. A read from data returns as many whole characters as the read count will permit starting at the beginning of the addr address (the end of the address has no effect) and sets the address to the null string at the end of the returned characters.

`errors` Writing to the errors file appends to the body of the dir/+Errors window, where dir is the directory currently named in the tag. The window is created if necessary, but not until text is actually written.

`event` When a window’s event file is open, changes to the window occur as always but the actions are also reported as messages to the reader of the file. Also, user actions with buttons 2 and 3 (other than chorged Cut and Paste, which behave normally) have no immediate effect on the window; it is expected that the program reading the event file will interpret them. The messages have a fixed format: a character indicating the origin or cause of the action, a character indicating the type of the action, four free-format blank-terminated decimal numbers, optional text, and a newline. The first and second numbers are the character addresses of the action, the third is a flag, and the final is a count of the characters in the optional text, which may itself contain newlines. The origin characters are
E for writes to the body or tag file, F for actions through the window’s other files, K for the keyboard, and M for the mouse. The type characters are D for text deleted from the body, d for text deleted from the tag, I for text inserted to the body, i for text inserted to the tag, L for a button 3 action in the body, l for a button 3 action in the tag, X for a button 2 action in the body, and x for a button 2 action in the tag.

If the relevant text has less than 256 characters, it is included in the message; otherwise it is elided, the fourth number is 0, and the program must read it from the data file if needed. No text is sent on a D or d message.

For D, d, I, and i the flag is always zero. For X and x, the flag is a bitwise OR (reported decimal) of the following: 1 if the text indicated is recognized as an acme built-in command; 2 if the text indicated is a null string that has a non-null expansion; if so, another complete message will follow describing the expansion exactly as if it had been indicated explicitly (its flag will always be 0); 8 if the command has an extra (chorded) argument; if so, two more complete messages will follow reporting the argument (with all numbers 0 except the character count) and where it originated, in the form of a fully-qualified button 3 style address.

For L and l, the flag is the bitwise OR of the following: 1 if acme can interpret the action without loading a new file; 2 if a second (post-expansion) message follows, analogous to that with X messages; 4 if the text is a file or window name (perhaps with address) rather than plain literal text.

For messages with the 1 bit on in the flag, writing the message back to the event file, but with the flag, count, and text omitted, will cause the action to be applied to the file exactly as it would have been if the event file had not been open.

**tag** holds contents of the window tag. It may be read at any byte offset. Text written to tag is always appended; the file offset is ignored.

**xdata**

The xdata file like data except that reads stop at the end address.

**SOURCE**

/sys/src/cmd/acme

**SEE ALSO**

rio(1), acme(1), cons(3).
NAME
  archfs - mount mkfs-style archive

SYNOPSIS
  archfs [-abc][ -m mtpt] archfile

DESCRIPTION
  Archfs mounts at mtpt (default /mnt/arch) a file system presenting the contents of an archive in
  the format produced by the -a flag to mkfs(8). The -a, -b, -c, and -C flags control the flag
  argument to the mount system call (see bind(2)) as in the mount command (see bind(1)).

SOURCE
  /sys/src/cmd/archfs.c

SEE ALSO
  mkfs(8)
NAME
cdfs, cddb - optical disc (CD, DVD, BD) track reader and writer file system

SYNOPSIS
cdfs [-d sddev] [-m mtpt]
grep aux/cddb /mnt/cd/ctl | rc
aux/cddb [-DTt] [-s server] query diskid ntracks track0id ...

DESCRIPTION
Cdfs serves a one and a half level directory mounted at mtpt (default /mnt/cd) that provides
access to the tracks on discs placed in the disc reader or writer named by sddev (default
/dev/sdD0, see sd(3)). Any MMC-compliant compact disc (CD), DVD, or Blu-ray disc (BD) drive
should work. On DVDs and BDs, access to data tracks only is implemented.

The top level directory contains one file per disc track. The files are named cNNN, where c is a
type character (a for audio tracks and d for data tracks) and NNN is the track number.

If the device can write discs and contains a writable disc, the top-level directory also contains an
empty directory wd and, for CDs only, an empty directory wa. Files created in these directories
appear in the top-level directory as new data or audio tracks, respectively, regardless of name.

At any time, any number of tracks may be open for reading or a single track may be open for writing.
Writing a disc track is a quasi-real-time operation: the disc writer should be kept saturated
with new data to avoid buffer underruns, but modern drives will be told to cope with underruns
transparently. To ensure saturation, copying from a file system stored on local disk or memory is
recommended.

To fixate a disc (close a recordable disc by writing its permanent table of contents), simply remove
the wa or wd directory. The directory removed selects whether the disc is fixated as an audio or
data disc; since each track carries its own type information, very few readers care which fixation
type was used. Rewritable discs do not require fixation.

The top level directory also contains a ctl file, into which control messages may be echoed. The
current control messages are:

format Format the rewritable disc (−RW or −RE) in the drive before initial use.
blank Blank the entire rewritable disc in the drive.
quickblank Blank only the table of contents on the rewritable disc in the drive.
eject Eject the disc in the drive.
ingest Ingest a disc into the drive.
speed kbps Set the reading and writing speed to use, in units of 1,000-bytes-per-second. A
value of best requests the optimal speed for the current drive and disc. CD 1x
speed is 154; DVD 1x speed is 1350; BD 1x speed is 4608. Drives may round
down the speed to one they support. To set reading and writing speeds sepa-
rately, prefix the speeds with read or write, as in speed read 8192 or
speed write 16384 write 8192. Note that most drives reset the reading
and writing speed each time a new disc is inserted.

Reading the ctl file yields information about the drive. If the drive contains an audio CD, the first
line will be an aux/cddb command that can be run to query an internet CD database to get a
table of contents. Subsequent lines contain the current and maximum reading and writing speeds.
Additional lines may further describe the current disc.

Aux/cddb takes 4 optional arguments. The −s option makes aux/cddb use server for the query
instead of freedb.freedb.org. The −D option causes the raw database response from the
server to be dumped to standard output. The −t option causes the time of each track to be
appended to the normal output. −T is like −t but prints a final line with the total time.

EXAMPLES
Backup to a BD−R disc:

9fs boot
cdfs
tar cf /mnt/cd/wd/x /n/boot

656
Copy the audio tracks from a CD:
   cdfs -d /dev/sd05
   mkdir /tmp/songs
   cp /mnt/cd/a* /tmp/songs

Copy the tracks onto a blank CD inserted in the drive, and then fixate the disk as an audio CD.
   cp /tmp/songs/* /mnt/cd/wa
   rm /mnt/cd/wa

SOURCE
/sys/src/cmd/cdfs

SEE ALSO
   sd(3), 9660srv (in dosrv(4)), mk9660(8)
   http://www.t10.org  optical disc interface standards

BUGS
   Fixating a BD–R disc records only the first track in the disc's TOC. Any other tracks are still there and their data accessible via sd(3). There's no need to fixate data discs, except to prevent adding new tracks.

   Closing a just–written DVD–R track can take minutes while the drive burns the unused part of the track reservation (for the whole disc). Thus only a single DVD–R track can be written on a DVD–R disc; use other media if you need more than one track per disc.

   There are too many combinations of optical media, each with unique quirks, approximately the cross–product of these tuples: (CD DVD– DVD+ BD), (single–layer dual–layer), (–ROM –R –RW).

   Only MMC–compliant disc readers and writers are supported, but it would be easy to add support for early CD writers if desired.
NAME
  cfs - cache file system

SYNOPSIS
  cfs −s [-dknrS] [−f partition]
  cfs −a netaddr [-dknrS] [−f partition] [mtpt]
  cfs −F srvfile [-dknrS] [−f partition] [mtpt]

DESCRIPTION
  Cfs is a user−level file server that caches data from remote files onto a local disk. It is normally
  started by the kernel at boot time, though users may start it manually. Cfs is interposed between
  the kernel and a network connection to a remote file server to improve the efficiency of access
  across slow network connections such as modem lines. On each open of a file cfs checks the con−
  sistency of cached information and discards any old information for that file.
  Cfs mounts onto mtpt (default /) after connecting to the file server.

The options are:

  a netaddr
     dial the destination netaddr to connect to a remote file server. Exclusive with −F.
  d turn on debugging.
  f partition
     use file partition as the cache disk partition.
  F srvfile
     open srvfile (often a file under /srv) to connect to a remote file server. Exclusive with −a.
  k keep cache contents even if they might have come from a different server. Cfs will obey −r
     even if −k is given.
  n mount the remote file server without authentication; often useful with −F.
  r reformat the cache disk partition.
  s the connection to the remote file server is on file descriptors 0 and 1.
  S turn on statistics gathering. A file called cfsctl at the root of the caching file system can be
     read to get statistics concerning number of calls/bytes on client and server sides and laten−
     cies.

All 9P messages except read, clone, and walk (see intro(5)) are passed through cfs unchanged
  to the remote server. If possible, a read is satisfied by cached data. Otherwise, the file server is
  queried for any missing data.

FILES
  /dev/sdC0/cache
     Default file used for storing cached data.

SOURCE
  /sys/src/cmd/cfs
NAME
cifs – Microsoft™ Windows network filesystem client

SYNOPSIS

DESCRIPTION
Cifs translates between Microsoft’s file-sharing protocol (a.k.a. CIFS or SMB) and 9P, allowing Plan9 clients to mount file systems (shares or trees in MS terminology) published by such servers.
The root of the mounted directory contains one subdirectory per share, always named in lower case, and a few virtual files of mixed case which give additional server, session, share, and user information. The arguments are:

-a auth-method Cifs authenticates using BNTLM by default, but alternative strategies may be selected using this option. Cifs eschews cleartext authentication, however it may be enabled with the plain auth method. The list of currently-supported methods is printed if no method name is supplied.

Windows server 2003 requires the BNTLMv2 method by default, though it can be configured to be more flexible.

-b Enable file ownership resolution in stat(2) calls. This requires an open and close per file and thus will slow cifs considerably; its use is not recommended.

-d CIFS packet debug.

-D 9P request debug.

-k keyparam lists extra parameters which will be passed to factotum(4) to select a specific key. The remote server’s domain is always included in the keyspec, under the assumption that all servers in a Windows domain share an authentication domain, thus cifs expects keys in factotum of the form:

    key proto=pass dom=THEIR-DOMAIN service=cifs
    user=MY-USERNAME !password=XYZZY

-m mntpnt set the mount point for the remote filesystem; the default is /n/ host.

-n called-name The CIFS protocol requires clients to know the NetBios name of the server they are attaching to, the lcalled-name. If this is not specified on the command line, cifs attempts to discover this name from the remote server. If this fails it will then try host, and finally it will try the name *SMBSERVER.

-s srvname post the service as /srv/srvname.

host The address of the remote server to connect to.

share A list of share names to attach on the remote server; if none is given, cifs will attempt to attach all shares published by the remote host.

Synthetic Files
Several synthetic files appear in the root of the mounted filesystem:

Shares Contains a list of the currently attached shares, with fields giving the share name, disk free space / capacity, the share type, and a descriptive comment from the server.

Connection Contains the username used for authentication, server’s called name, server’s domain, server’s OS, the time slip between the local host and the server, the Maximum Transfer Unit (MTU) the server requested, and optionally a flag indicating only guest access has been granted. The second line contains a list of capabilities offered by the server which is mainly of use for debugging cifs.

Users Each line contains a user’s name, the user’s full name, and a descriptive comment.
Groups
   Each line gives a group's name, and a list of the names of the users who are
   members of that group.

Sessions
   Lists the users authenticated, the client machine's NetBios name or IP address,
   the time since the connection was established, and the time for which the con-
   nection has been idle.

Domains
   One line per domain giving the domain name and a descriptive comment.

Workstations
   One line per domain giving the domain name and a descriptive comment, the
   version number of the OS it is running, and comma-separated list of flags giv-
   ing the features of that OS.

Dfsroot
   Top level DFS routing giving the DFS link type, time to live of the data, prox-
   imity of the server, the Netbios or DNS name and a physical path or a machine
   that this maps to.

   DNS paths are usually assigned dynamically as a form of load balancing.

SOURCE
   /sys/src/cmd/cifs

SEE ALSO
   factotum(4), aquarela(8)

BUGS
   NetApp Filer compatibility has not yet been tested; there may not be any.
   DFS support is unfinished.
   Kerberos authentication is unfinished.
   NetBios name resolution is not supported, though it is now rarely used.
   Cifs has only been tested against aquarela(8), Windows 95, NT4.0sp6, Windows server 2003,
   WinXP pro, Samba 3.0, and Samba 2.0 (Pluto VideoSpace). No support is attempted for servers
   predating NT 4.0.
NAME
consolefs, C, clg – file system for console access

SYNOPSIS
  aux/consolefs [-m mntpt] [-c consoledb]
  C system
  aux/clog console log system

DESCRIPTION
To ease administration of multiple machines one might attach many serial console lines to a single
computer. Consolefs is a file system that lets multiple users simultaneously access these console
lines. The consoles and permissions to access them are defined in the file consoledb (default
/lib/ndb/consoledb). The format of consoledb is the same as that of other /lib/ndb
files, ndb(6). Consoles are defined by entries of the form:
  console=dirty dev=/dev/eia205
  uid=bignose
  gid=support
  speed=56200
  cronly=

Each console/dev pair represents the name of a console and the device associated with it. Consolefs presents a single level directory with up to three files per console: console, consolectl, and consolestat. Writes of console are equivalent to writes of dev and reads and writes of
consolectl and consolestat are equivalent to reads and writes of devctl and devstat respectively. Consolectl and consolestat will not exist if the underlying dev does not provide them. Consolefs broadcasts anything it reads from dev to all readers of console. Therefore, many users can con(1) to a console, see all output, and enter commands.

The cronly= attribute causes newlines typed by the user to be sent to the console as returns. The
speed=x attribute/value pair specifies a bit rate for the console. The default is 9600 baud. The
openondemand= attribute causes the console device (dev) to be opened only when the correspond-
ing mntpt/console file is open.

Access to the console is controlled by the uid and gid attributes/value pairs. The uid values are
user account names. The gid values are the names of groups defined in consolefs by entries of the
form:
  group=support
   uid=bob
   uid=carol
   uid=ted
   uid=alice

Groups are used to avoid excessive typing. Using gid=x is equivalent to including a uid=y for each
user y that is a member of x.

To keep users from inadvertently interfering with one another, notification is broadcast to all read-
ers whenever a user opens or closes name. For example, if user boris opens a console that users
vlad and barney have already opened, all will read the message:
  [+boris, vlad, barney]
If vlad then closes, boris and barney will read:
  [-vlad, boris, barney]

Consolefs posts the client end of its 9P channel in /srv/consolefs and mounts this locally in
mntpt (default /mnt/consoles); remote clients must mount (see bind(1)) this file to see the
consoles.

The rc(1) script C automates this procedure. It uses import(4) to connect to /mnt/consoles
on the machine connected to all the consoles, then uses con(1) to connect to the console of the
machine system. The script must be edited at installation by the local administration to identify the
system that holds /mnt/consoles.
Aux/clog opens the file console and writes every line read from it, prefixed by the ASCII time to the file log.

An example of 2 consoles complete with console logging is:

```bash
% cat /lib/ndb/consoledb
  group=sys
    uid=glenda
  console=bootes dev=/dev/eia0 gid=sys
  console=fornax dev=/dev/eial gid=sys
% aux/consolefs
% ls -p /mnt/consoles
  bootes
  bootesctl
  fornax
  fornaxctl
% clog /mnt/consoles/fornax /sys/log/fornax &
% clog /mnt/consoles/bootes /sys/log/bootes &
```

The console server's default name space must mount the consoles for C to import. This can be arranged by adding

```bash
mount /srv/consoles /mnt/consoles
to /lib/namespace.$sysname.
```

**FILES**

- `/srv/consoles` Client end of pipe to server.
- `/mnt/consoles` Default mount point.
- `/lib/ndb/consoledb` Default user database.

**SOURCE**

- `/sys/src/cmd/aux/consolefs.c`
- `/rc/bin/C`
- `/sys/src/cmd/aux/clog.c`

**BUGS**

Changing the gid's or uid's while consolefs is running is detected by consolefs. However, to add new consoles one must restart consolefs.
NAME
cvsfs – cvs repository viewer

SYNOPSIS
cvsfs [−Ddabv] [−s srv] [−k keyp] [−m mntpnt] [:method:[user@]]host: /root module

DESCRIPTION
cvsfs creates a read–only view of a remote CVS repository module. Two views are presented onto
the repository, one indexed by date – similar to venti(1)’s dump filesystem, the other indexed by
tag name. A special tag directory HEAD contains the most up–to–date view of the repository. The
file ChangeLog at the top level contains an auto–generated canonical history of the changes to
the module.
File metadata is only filled in when the file is first read. A file’s owner is the author of the last
change, and its group is the name of the locker of the file or unlocked if none. Empty directories
are not stored by CVS and thus are not listed.
The options are:
−v Display server version string and some stats at startup. (See BUGS below)
−a −b The same after and before options as documented in mount(1).
−D 9P request debug.
−d CVS protocol debug.
−m mntpnt
Set the mount point for the remote filesystem; the default is /n/cvs.
−k keyparam Extra parameters which will be passed to factotum(4) to remove key ambiguity.
−s srvname
Set the mount point for the published /srv file. The default is not to publish a /srv file.

method
CVS allows several methods for attaching to servers. Currently only ext and pserver
are supported. The ext method attempts to use ssh(1), cpu(1) and rx (see con(1)) in that
order.

username
The user name to attach as.

host
The name of the machine to attach to.

cvsroot
The path to the root of the CVS filesystem on the remote machine.

module
The name of the module on the remote server

BUGS
Not actually present.
No support for Kerberos GSSAPI authentication – the gserver method.
All files are read (checked out) on demand. Files not accessed for 10 mins have their contents
flushed. The CVS server remembers which files have been checked out and refuses to resend
these. As a result cvsfs must disconnect & reconnect on each read–idle–reread cycle. This is handled
by cvsfs but may cause a delay on open(2).
cvsfs relies on parsing CVS r10g data to build its knowledge of the tree, This data is designed to
be human readable and is thus liable to change between CVS server versions, though the authors
of CVS do endeavour not to change it unnecessarily.
The −v option causes a CVS ver request to be sent. This can cause some servers (e.g. those used
by sourceforge) to abort with the POSIX error “Terminated with fatal signal 11”; do not use the −v
option with such servers.
Cvsfs has currently (February 2005) been tested with cvs V1.12.9 and V1.11.1p1.
NAME
cwfs - cached--worm file server, dump

SYNOPSIS
cwfs [ -cf ] [ -a announce-string ] ... [ -m device-map ] config--device

DESCRIPTION
Cwfs is a cached--worm file server that runs as a user--mode program and can maintain file systems created by fs(4), the original Plan 9 file server that had its own kernel and operated a stand--alone system with disks and optical--disc jukebox attached. Unlike fs(4), which could only accept 9P connections over IL/IPv4 on EtHERnets (or over Datakit and Cyclones, long ago), cwfs accepts 9P connections on any network medium and protocol that it can announce on, by default TCP (over IPv4 or IPv6). Given suitable 9P clients, one could even run 9P over aan(8) or tls(3).

The stock cwfs implements a 16K file system block size and 32--bit disk addresses, in order to be compatible with some existing file systems, notably emelie's. These parameters can be changed by recompilation.

Cwfs expects to find the configuration block on config--device.

Options are:
- a announce on announce-string instead of tcp!*!9fs.
- c use a newer, faster, and incompatible cache--device layout. To convert an old file system's cache to the new layout, dump the file system, note the last superblock number, halt cwfs, restart cwfs with -cf, recover the file system, and start cwfs with -c thereafter.
- f enter the file server's configuration mode before starting normal operation.
- m the file device-map contains a simple device name (e.g., w9) and a replacement per line. The device name in the usual filsys notation of fsconfig(8). The replacement can be the name of an existing file (which cwfs will not grow) or another such device name. For example, the file

  w0 /tmp/w0
  h1 w2

would map accesses to device w0 to existing file /tmp/w0 and accesses to device h1 to device w2, if no file named w2 exists.

The file server normally requires all users except none to provide authentication tickets on each attach(5). This can be disabled using the noauth configuration command (see fsconfig(8)).

The group numbered 9999, normally called noworld, is special on the file server. Any user belonging to that group has attenuated access privileges. Specifically, when checking such a user's access to files, the file's permission bits are first ANDed with 0770 for normal files or 0771 for directories. The effect is to deny world access permissions to noworld users, except when walking directories.

The user none is always allowed to attach to emelie without authentication but has minimal permissions.

Emelie maintains three file systems on a combination of disks and write--once--read--many (WORM) magneto--optical disks.

other

is a simple disk--based file system similar to kfs(4).

main

is a worm--based file system with a disk--based look--aside cache. The disk cache holds modified worm blocks to overcome the write--once property of the worm. The cache also holds recently accessed non--modified blocks to speed up the effective access time of the worm. Occasionally (usually daily at 5AM) the modified blocks in the disk cache are dumped. At this time, traffic to the file system is halted and the modified blocks are relabeled to the unwritten portion of the worm. After the dump, the file system traffic is continued and the relabeled blocks are copied to the worm by a background process.

dump

Each time the main file system is dumped, its root is appended to a subdirectory of the dump file system. Since the dump file system is not mirrored with a disk cache, it is read--only. The name of the newly added root is created from the date of the dump:
/yyyymmddds. Here yyyy is the full year, mm is the month number, dd is the day number and s is a sequence number if more than one dump is done in a day. For the first dump, s is null. For the subsequent dumps s is 1, 2, 3, etc.

The root of the main file system that is frozen on the first dump of March 1, 1992 will be named /1992/0301/ in the dump file system.

Changes from fs(4)

fs(4)'s IP configuration is ignored and the underlying system's is used.
Various other fs(4) commands have been omitted since they (or equivalents) can now be executed directly on the underlying CPU server, notably date and passwd (see auth/wrkey).
fs(4)'s device names h for IDE disks and m for Marvell SATA disks are not supported; use -m to map wren devices to appropriate names under /dev/sd*.
The file server kernel seems to have scanned PCI buses in reverse order from the other Plan 9 kernels, so systems with multiple SCSI cards may find controller numbering reversed. -m can be used to compensate for this if you don't want to change filsys declarations.
The file server kernel's config field in NVRAM was overloaded in recent times to hold a seastore(1) key for the CPU hostowner. Since cwfs runs on a CPU kernel, the location of its configuration block must be supplied on the command line.
Disk labels are now implemented for 1 devices. At the first access of a side, cwfs will attempt to read the label and verify that it has the correct side number and byte order; if either is wrong, it will issue a warning. If the label cannot be read, cwfs will attempt to write a new label.

EXAMPLES

Place the root of the dump file system on /n/dump and show the modified times of the MIPS C compiler over all dumps in February, 1992:

cwfs w0
9fs dump
ls -l /n/dump/1992/02??/mips/bin/vc

To get only one line of output for each version of the compiler:

ls -lp /n/dump/1992/02??/mips/bin/vc | uniq

SOURCE

/sys/src/cmd/cwfs

SEE ALSO

yesterday(1), fs(3), sd(3), fossil(4), fs(4), srv(4), fs(8), fsconfig(8)
Ken Thompson, Geoff Collyer, “The 64-bit Standalone Plan 9 File Server”

BUGS

For the moment, the file server serves both the old (9P1) and new (9P2000) versions of 9P, deciding which to serve by sniffing the first packet on each connection.
File system block size and disk address size (32- or 64-bit) are fixed at compilation time, and this is not easily changed.
Cwfs is probably not the right choice of file server for new file systems. It's intended to cope with existing file systems on optical jukeboxes or images thereof.
NAME
dossrv, 9660srv, a:, b:, c:, d:, 9fat:, dosmnt, eject - DOS and ISO9660 file systems

SYNOPSIS
dossrv [-rsv] [-f file] [ service ]
9660srv [-9Jsv] [-c clusters] [-f file] [ service ]
a:
b:
c:
9fat:
dosmnt n mtpt
eject [n]

DESCRIPTION
Dossrv is a file server that interprets DOS file systems. A single instance of dossrv can provide access to multiple DOS disks simultaneously.

Dossrv posts a file descriptor named service (default dos) in the /srv directory. To access the DOS file system on a device, use mount with the spec argument (see bind(1)) the name of the file holding raw DOS file system, typically the disk. If spec is undefined in the mount, dossrv will use file as the default name for the device holding the DOS system.

Normally dossrv creates a pipe to act as the communications channel between itself and its clients. The -s flag instructs dossrv to use its standard input and output instead. The kernels use this option if they are booting from a DOS disk. This flag also prevents the creation of an explicit service file in /srv.

The -v flag causes verbose output for debugging, while the -r flag makes the file system read-only.

The shell script a: contains

    umount /n/a: >[2] /dev/null
    mount -c /srv/dos /n/a: /dev/fd0disk

and is therefore a shorthand for mounting a floppy disk in drive A. The scripts b: and dosmnt are similar, mounting the second floppy disk and the nth non-floppy DOS partition, respectively. C: and d: call dosmnt in an attempt to name the drives in the same order that Microsoft operating systems do. 9fat: provides access to the FAT component of the Plan 9 partition (see prep(8)).

The file attribute flags used by the DOS file system do not map directly to those used by Plan 9. Since there is no concept of user or group, permission changes via wstat (see stat(2)) will fail unless the same (read, write, execute) permissions are specified for user, group, and other. For example, removing write permission in Plan 9 corresponds to setting the read-only attribute in the DOS file system. Most of the other DOS attributes are not accessible.

Setting the exclusive use flag (DMEXCL) in Plan 9 corresponds to setting the system use attribute in the DOS file system. Such files are not actually restricted to exclusive use, but do merit special treatment that helps in the creation of boot disks: when dossrv allocates a new block for such a file (caused, say, by a write that fills the file's last allocated block), it succeeds only if it can arrange for the file to be stored contiguously on disk.

Since other operating systems do not guarantee that system files are laid out contiguously, the DMAPPEND mode bit is set in file stat information only when the file is currently contiguous. Attempts to set the DMAPPEND mode bit explicitly will cause dossrv to try to make the file contiguous, succeeding only if this is possible.

9660srv is similar to dossrv in specification, except that it interprets ISO9660 CD-ROM file systems instead of DOS file systems. Some CDs contain multiple directory trees describing the same set of files. 9660srv’s first choice in such a case is a standard ISO9660 tree with Plan 9 system use fields; the second choice is a Microsoft “Joliet” tree, which allows long file names and Unicode characters; the third choice is a standard ISO9660 or High Sierra tree. The -9 flag causes 9660srv
to ignore the Plan 9 system use fields, while the –J flag causes it to ignore the Joliet tree. The –c option sets the size of the RAM cache to clusters clusters of 128KB. The default clusters is 16, but a value of 5600 will cache an entire CD incrementally.

If the floppy drive has an ejection motor, eject will spit out the floppy from drive n, default 0.

EXAMPLE
Mount a floppy disk with a DOS file system on it.

    a:

SEE ALSO
kfs(4)

SOURCE
/sys/src/cmd/dossrv
/sys/src/cmd/9660srv
/rc/bin/eject

BUGS
The overloading of the semantics of the DMEXCL and DMAPPEND bits can be confusing.
NAME
  execnet - network interface to program execution

SYNOPSIS
  execnet [−n name][netdir]

DESCRIPTION
  Execnet presents a network protocol directory (see, for example, ip(3)) called netdir / name (default /net/exec).
  Once the protocol directory exists, dialing (see dial(2)) strings of the form name ! cmd will connect to a newly executed instance of cmd.

EXAMPLE
  Execnet can be used to connect to instances of u9fs(4) running on other hosts:
  
g% execnet
  g% srv −m 'exec!ssh ny start−u9fs' ny /n/ny
  
  This example assumes that the remote command start−u9fs executed on ny will start u9fs appropriately. For example, it might be:
  
  ny% cat start−u9fs
  #!/bin/sh
  
  u9fs −na none −u $USER -l $HOME/tmp/u9fs.log
  
  See the u9fs(4) man page for more information.

SOURCE
  /sys/src/cmd/execnet

SEE ALSO
  dial(2), ip(3), u9fs(4)

BUGS
  Almost certainly: execnet has only been tested as in the example shown.
NAME
exportfs, srvfs - network file server plumbing

SYNOPSIS
exportfs [ options ]

DESCRIPTION
Exportfs is a user level file server that allows Plan 9 compute servers, rather than file servers, to export portions of a name space across networks. The service is started either by the cpu(1) command or by a network listener process. An initial protocol establishes a root directory for the exported name space. The connection to exportfs is then mounted, typically on /mnt/term. Exportfs then acts as a relay file server: operations in the imported file tree are executed on the remote server and the results returned. This gives the appearance of exporting a name space from a remote machine into a local file tree.

The options are:

-A address Use the network address to announce aam(8) connections, if requested by the initial protocol.

-a Authenticate the user with the p9any protocol before running the regular exportfs session: used when exportfs is invoked to handle an incoming network connection. Exportfs creates a new name space for each connection, using /lib/namespace by default (see namespace(6)).

-B address Dial address, authenticate as a p9any client, and then serve that network connection. Requires setting the root of the name space with -r or -s. The remote system should run import -B to handle the call. See import(4) for an example.

-d -f dbgfile
Log all 9P traffic to dbgfile (default /tmp/exportdb).

-e 'enc auth'
Set the encryption and authentication algorithms to use for encrypting the wire traffic (see ssl(3)). The defaults are rc4_256 and sha1.

-m msize Set the maximum message size that exportfs should offer to send (see version(5)); this helps tunnel 9P connections to avoid unnecessary fragmentation.

-N nsfile Serve the name space described by nsfile.

-n Disallow mounts by user none.

-P patternfile
Restrict the set of exported files. Patternfile contains one regular expression per line, to be matched against path names relative to the current working directory and starting with .//. For a file to be exported, all lines with a prefix + must match and all those with prefix - must not match.

-R Make the served name space read only.

-r root Bypass the initial protocol, serving the name space rooted at root.

-S service bypass the initial protocol, serving the result of mounting service. A separate mount is used for each attach(5) message, to correctly handle servers in which each mount corresponds to a different client e.g., (rio(4)).

-s equivalent to -r /; kept for compatibility.

The cpu command uses exportfs to serve device files in the terminal. The import(4) command calls exportfs on a remote machine, permitting users to access arbitrary pieces of name space on other systems.

Because the kernel disallows reads and writes on mounted pipes (as might be found in /srv), exportfs calls itself (with appropriate -m and -S options) to simulate reads and writes on such files.
Srvfs invokes exportprog (default /bin/exportfs) to create a mountable file system from a name space and posts it at /srv/name, which is created with mode perm (default 0600). The name space is the directory tree rooted at path. The –d, –P, and –R options, if present, are relayed to exportprog.

EXAMPLES
To export the archive of one user for one month, except for secrets,

    cd /n/dump
    echo '+ ^\(/2003(\10..(/usr(/glenda/?)?))??\)' > /tmp/pattern
    echo '- \.(aes|pgp)$' >> /tmp/pattern
    exportfs −P /tmp/pattern

Use srvfs to enable mounting of an FTP file system (see ftpfs(4)) in several windows, or to publish a /proc (see proc(3)) with a broken process so a remote person may debug the program:

    srvfs ftp /n/ftp
    srvfs broke /mnt/term/proc

Use srvfs to obtain a copy of a service to be manipulated directly by a user program like nfsserver(8):

    srvfs nfs.boot /srv/boot
    aux/nfsserver −f /srv/nfs.boot

Use srvfs to spy on all accesses to a particular subtree:

    srvfs −d spy /
    tail −f /tmp/exportdb &
    mount /srv/spy /n/spy
    cd /n/spy; ls

SOURCE
/sys/src/cmd/exportfs
/sys/src/cmd/srvfs.c

SEE ALSO
dial(2), import(4), aan(8), listen(8)
NAME
ext2srv - ext2 file system

SYNOPSIS
ext2srv [−vrs][−f file][−p passwd][−g group][ service ]

DESCRIPTION
Ext2srv is a file server that interprets the Linux Second Extended File System. A single instance of ext2srv can provide access to multiple ext2 partitions simultaneously.

Ext2srv posts a file descriptor named service (default ext2) in the /srv directory. To access an ext2 file system on a device, use mount with the spec argument (see bind(1)) the name of the file holding the raw ext2 file system, typically the disk or partition. If spec is undefined in the mount, ext2srv will use file as the default name for the device holding the file system.

Normally ext2srv creates a pipe to act as the communications channel between itself and its clients. The −s flag instructs ext2srv to use its standard input and output instead. This flag also prevents the creation of an explicit service file in /srv.

The −v flag causes verbose output for debugging, while the −r flag (recommended) makes the file system read-only. The optional −p and −g flags specify Unix-format password (respectively group) files that give the mapping between the numeric user- and group-ID numbers in the ext2 file system and the strings reported by Plan 9 status inquiries.

There is no authentication or permission checking. Anyone who can access the ext2 file system will have full access to all its files, including write access if ext2srv is not started with the −r flag, irrespective of file ownership and permission flags.

Some file system state is cached in memory, and may be flushed only when the file system is unmounted. Therefore if ext2srv is stopped or the machine is rebooted while an ext2 file system is still mounted, the superblock on the device will have been marked ‘not valid’ (unless the −r flag was used), and a fsck will be required before that file system may be mounted again.

BUGS
There is no authentication or permission checking. The implementation has not tracked any changes to the ext2 specification since it was written. There may be other bugs. It is advisable to use ext2srv in read-only mode whenever possible.

AUTHOR
Bodet Laurent (bl@mime.univ-paris8.fr), with later updates by Russ Cox and Richard Miller.
NAME
factotum, fgui - authentication agent

SYNOPSIS
auth/factotum [-DdknpuS][-a asaddr][-s srvname][-m mtpt]
auth/factotum -g attribute=value ... attribute?
auth/fgui

DESCRIPTION
Factotum is a user-level file system that acts as the authentication agent for a user. It does so by managing a set of keys. A key is a collection of information used to authenticate a particular action. Stored as a list of attribute=value pairs, a key typically contains a user, an authentication domain, a protocol, and some secret data.

Factotum presents a two-level directory. The first level contains a single directory factotum, which in turn contains:

rpc each open represents a new private channel to factotum
proto when read lists the protocols available
confirm for confirming the use of key
needkey allows external programs to control the addition of new keys
log a log of actions
ctl for maintaining keys; when read, it returns a list of keys. For secret attributes, only the attribute name follow by a ? is returned.

In any authentication, the caller typically acts as a client and the callee as a server. The server determines the authentication domain, sometimes after a negotiation with the client. Authentication always requires the client to prove its identity to the server. Under some protocols, the authentication is mutual. Proof is accomplished using secret information kept by factotum in conjunction with a cryptographic protocol.

Factotum can act in the role of client for any process possessing the same user id as it. For select protocols such as p9sk1 it can also act as a client for other processes provided its user id may speak for the other process' user id (see authsrv(6)). Factotum can act in the role of server for any process.

Factotum's structure is independent of any particular authentication protocol. Factotum supports the following protocols:
p9any a metaprotocol used to negotiate which actual protocol to use.
p9sk1 a Plan 9 shared key protocol described in authsrv(6)'s "File Service" section.
p9sk2 a variant of p9sk1 described in authsrv(6)'s "Remote Execution" section.
p9cr a Plan 9 protocol that can use either p9sk1 keys or SecureID tokens.
apop the challenge/response protocol used by POP3 mail servers.
cram the challenge/response protocol also used by POP3 mail servers.
chap the challenge/response protocols used by PPP and PPTP.
mschap a proprietary Microsoft protocol also used by PPP and PPTP.
rsa RSA public key decryption, used by SSH and TLS.
pass passwords in the clear.
vnc vnc(1)'s challenge/response.
wep WEP passwords for wireless ethernet cards.

The options are:
-a supplies the address of the authentication server to use. Without this option, it will attempt to find an authentication server by querying the connection server, the file <mtpt>/ndb, and finally the network database in /lib/ndb.
-m specifies the mount point to use, by default /mnt.
-s specifies the service name to use. Without this option, factotum does not create a service file in /srv.
-D turns on 9P tracing, written to standard error.
-d turns on debugging, written to standard error.
-g causes the agent to prompt for the key, write it to the ct1 file, and exit. The agent will prompt for values for any of the attributes ending with a question mark (?) and will append all the supplied attribute=value pairs. See the section on key templates below.
-n don’t look for a secstore.
-S indicates that the agent is running on a CPU server. On starting, it will attempt to get a p9sk1 key from NVRAM using readvram (see authsrv(2)), prompting for anything it needs. It will never subsequently prompt for a key that it doesn’t have. This option is typically used by the kernel at boot time.
-k causes the NVRAM to be written. It is only valid with the -S option. This option is typically used by the kernel at boot time.
-u causes the agent to prompt for user id and writes it to /dev/hostowner. It is mutually exclusive with -k and -S. This option is typically used by the kernel at boot time.
-p causes the agent not to mark itself ‘private’ via proc(3), so that it can be debugged. It is implied by -d.

Fgui is a graphic user interface for confirming key usage and entering new keys. It hides the window in which it starts and waits reading requests from confirm and needkey. For each request, it unhides itself and waits for user input. See the sections on key confirmation and key prompting below.

Key Tuples
A key tuple is a space delimited list of attribute=value pairs. An attribute whose name begins with an exclamation point (!) does not appear when reading the ct1 file. The required attributes depend on the authentication protocol.

P9sk1, p9sk2, and p9cr all require a key with proto=p9sk1, a dom attribute identifying the authentication domain, a user name valid in that domain, and either a !password or !hex attribute specifying the password or hexadecimal secret to be used. Here is an example:

proto=p9sk1 dom=avayalabs.com user=presotto !password=lucent

Apop, cram, chap, and mschap, require a key with a proto attribute whose value matches the protocol, in addition to server, user, and !password attributes; e.g.

proto=apop server=mit.edu user=rsc !password=nerdsRus

Vnc is similar but does not require a user attribute.

Pass requires a key with proto=pass in addition to user and !password attributes; e.g.

proto=pass user=tb !password=does.it.matter

Rsa requires a key with proto=rsa in addition to all the hex attributes defining an RSA key: ek, n, !p, !q, !kp, !kq, !c2, and !dk. By convention, programs using the RSA protocol also require a service attribute set to ssh, sshserve, or tls.

Wep requires a key1, key2, or key3 set to the password to be used. Starting the protocol causes factotum to configure the wireless ethernet card #1/ether0 for WEP encryption with the given password.

All keys can have additional attributes that act either as comments or as selectors to distinguish them in the auth(2) library calls.

The factotum owner can use any key stored by factotum. Any key may have one or more owner attributes listing the users who can use the key as though they were the owner. For example, the TLS and SSH host keys on a server often have an attribute owner=* to allow any user (and in particular, none) to run the TLS or SSH server-side protocol.

Any key may have a role attribute for restricting how it can be used. If this attribute is missing, the key can be used in any role. The possible values are:

client
for authenticating outbound calls

server
for authenticating inbound calls
speakfor
for authenticating processes whose user id does not match factotum’s.

If a key has a disabled attribute (with any value), the key is not used during any protocols. Factotum automatically marks keys with disabled=by_factotum when they fail during certain authentication protocols (in particular, the Plan 9 ones).

Whenever factotum runs as a server, it must have a p9sk1 key in order to communicate with the authentication server for validating passwords and challenge/responses of other users.

**Key Templates**
Key templates are used by routines that interface to factotum such as auth_proxy and auth_challenge (see auth(2)) to specify which key and protocol to use for an authentication. Like a key tuple, a key template is also a list of attribute=value pairs. It must specify at least the protocol and enough other attributes to uniquely identify a key, or set of keys, to use. The keys chosen are those that match all the attributes specified in the template. The possible attribute/value formats are:

- `attr=val` The attribute `attr` must exist in the key and its value must exactly match `val`
- `attr?` The attribute `attr` must exist in the key but its value doesn’t matter.
- `attr` The attribute `attr` must exist in the key with a null value

Key templates are also used by factotum to request a key either via an RPC error or via the needkey interface. The possible attribute/value formats are:

- `attr=val` This pair must remain unchanged
- `attr?` This attribute needs a value
- `attr` The pair must remain unchanged

**Control and Key Management**
A number of messages can be written to the control file. The messages are:

- `key attribute=value-list` add a new key. This will replace any old key whose public, i.e. non ! attributes, match.
- `delkey attribute=value-list` delete a key whose attributes match those given.
- `debug` toggle debugging on and off, i.e., the debugging also turned on by the –d option.

By default when factotum starts it looks for a secstore(1) account on $auth for the user and, if one exists, prompts for a secstore password in order to fetch the file factotum, which should contain control file commands. An example would be:

```
key dom=x.com proto=p9sk1 user=boyd !hex=26E522ADE2BBB2A229
key proto=rsa service=ssh size=1024 ek=3B !dk=...
```

where the first line sets a password for challenge/response authentication, strong against dictionary attack by being a long random string, and the second line sets a public/private keypair for ssh authentication, generated by ssh_genkey (see ssh(1)).

**Confirming key use**
The confirm file provides a connection from factotum to a confirmation server, normally the program auth/fgui. Whenever a key with the confirm attribute is used, factotum requires confirmation of its use. If no process has confirm opened, use of the key will be denied. However, if the file is opened a request can be read from it with the following format:

```
confirm tag=tagno <key template>
```

The reply, written back to confirm, consists of string:

```
tag=tagno answer=xxx
```

If `xxx` is the string yes then the use is confirmed and the authentication will proceed. Otherwise, it fails.

Confirm is exclusive open and can only be opened by a process with the same user id as factotum.
Prompting for keys
The needkey file provides a connection from factotum to a key server, normally the program auth/gui. Whenever factotum needs a new key, it first checks to see if needkey is opened. If it isn’t, it returns an error to its client. If the file is opened a request can be read from it with the following format:

```
needkey tag=tagno <key template>
```

It is up to the reader to then query the user for any missing fields, write the key tuple into the ctl file, and then reply by writing into the needkey file the string:

```
tag=tagno
```

Needkey is exclusive open and can only be opened by a process with the same user id as factotum.

The RPC Protocol
Authentication is performed by

1) opening rpc
2) setting up the protocol and key to be used (see the start RPC below),
3) shuttling messages back and forth between factotum and the other party (see the read and write RPC’s until done
4) if successful, reading back an AuthInfo structure (see authsrv(2)).

The RPC protocol is normally embodied by one of the routines in auth(2). We describe it here should anyone want to extend the library.

An RPC consists of writing a request message to rpc followed by reading a reply message back. RPC’s are strictly ordered; requests and replies of different RPC’s cannot be interleaved. Messages consist of a verb, a single space, and data. The data format depends on the verb. The request verbs are:

```
start attribute-value-list
```

start a new authentication. Attribute-value-pair-list must include a proto attribute, a role attribute with value client or server, and enough other attributes to uniquely identify a key to use. A start RPC is required before any others. The possible replies are:

```
ok start succeeded.
```

```
error string
```

where string is the reason.

```
read get data from factotum to send to the other party. The possible replies are:
```

```
ok read succeeded, this is zero length message.
```

```
ok data
```

read succeeded, the data follows the space and is unformatted.

```
done authentication has succeeded, no further RPC’s are necessary
```

```
done haveai
```

authentication has succeeded, an AuthInfo structure (see auth(2)) can be retrieved with an authinfo RPC.

```
phase string
```

its not your turn to read, get some data from the other party and return it with a write RPC.

```
error string
```

authentication failed, string is the reason.

```
protocol not started
```

a start RPC needs to precede reads and writes

```
needkey attribute-value-list
```

a key matching the argument is needed. This argument may be passed as an argument to factotum –g in order to prompt for a key. After that, the authentication
may proceed, i.e., the read restarted.

**write** *data*
send data from the other party to *factotum*. The possible replies are:
**ok** the write succeeded
**needkey** *attribute-value-list*
see above
**toosmall** *n*
the write is too short, get more data from the other party and retry the write. *n*
specifies the maximun total number of bytes.

**phase** *string*
its not your turn to write, get some data from *factotum* first.
**done** see above
**done haveai**
see above

**authinfo**
retrieve the AuthInfo structure. The possible replies are:
**ok** *data*
*data* is a marshaled form of the AuthInfo structure.
**error** *string*
where *string* is the reason for the error.

**attr** retrieve the attributes used in the *start* RPC. The possible replies are:
**ok** *attribute-value-list*
**error** *string*
where *string* is the reason for the error.

**SOURCE**
/sys/src/cmd/auth/factotum
NAME
  flashfs - journalling file system for flash memory

SYNOPSIS
  aux/flashfs [ -Dr ] [ -n nsect ] [ -z sectsize ] [ -f file ] [ -m mountpoint ]

DESCRIPTION
  Flashfs interprets the journal-based file system created by mkflashfs(8) and stored in file (default /dev/flash/fs) so that it can be mounted into a Plan 9 file system. Flashfs is typically used to create a stand alone file system from a small persistent storage device, such as an erasable flash memory. It does not authenticate its clients and assumes each group has a single member with the same name.

  The -s option causes flashfs to post its channel on #s/flashfs. Flashfs mounts itself on mountpoint (default /n/bzrxr). The -D option turns on 9P debugging output. The -x option makes the file system read-only.

  The files and directory structure are divided into sectsize (default 4096) byte blocks. Larger blocks make large files more compact but take longer to access. Supplying the -n option forces file to contain exactly nsect sectors.

SOURCE
  /sys/src/cmd/aux/flashfs

SEE ALSO
  paqfs(4), sacfs(4), mkflashfs(8)
NAME
fossil, flchk, flfmt - archival file server

SYNOPSIS
fossil/fossil [-Dt][-c cmd]... [-F file][-m free-memory%]
fossil/flchk [-f][-c ncache][-h host] file
fossil/fflmnt [-y][-b blocksize][-h host][-l label][-v score] file
fossil/conf[-w] file [config]
fossil/last file

DESCRIPTION
Fossil is the main file system for Plan 9. Unlike the Plan 9 file servers of old, fossil is a collection of
user-space programs that run on a standard Plan 9 kernel. The name of the main fossil file server
at Murray Hill is pie. The Plan 9 distribution file server, sources, is also a fossil server.
Fossil is structured as a magnetic disk write buffer optionally backed by a Venti server for archival
storage. It serves the Plan 9 protocol via TCP. A fossil file server conventionally presents three
trees in the root directory of each file system: active, archive, and snapshot. /active is the
root of a conventional file system whose blocks are stored in a disk file. In a typical configuration,
the file server periodically marks the entire file system copy-on-write, effectively taking a
snapshot of the file system at that moment. This snapshot is made available in a name created
from the date and time of the snapshot: /snapshot/yyyy/mmdd/hhmm where yyyy is the full
year, mm is the month number, dd is the day number, hh is the hour, and mm is the minute.
The snapshots in /snapshot are ephemeral: eventually they are deleted to reclaim the disk space
they occupy. Long-lasting snapshots stored on a Venti server are kept in /archive and also
named from the date (though not the time) of the snapshot: /archive/yyyy/mmddds, where
yyyy, mm, and dd are year, month, and day as before, and s is a sequence number if more than
one archival snapshot is done in a day. For the first snapshot, s is null. For the subsequent snap-
shots, s is .1, .2, .3, etc. The root of the main file system that is frozen for the first archival
snapshot of December 15, 2002 will be named /archive/2002/1215/.
The attach name used in mount (see bind(1), bind(2) and attach(5)) selects a file system to be
served and optionally a subtree, in the format fs/dir]. An empty attach name selects
main/active.
Fossil normally requires all users except none to provide authentication tickets on each attach(5).
To keep just anyone from connecting, none is only allowed to attach after another user has suc-
cessfully attached on the same connection. The other user effectively acts as a chaperone for
none. Authentication can be disabled using the −A flag to open or srv (see fossilcons(8)).
The groups called noworld and write are special on the file server. Any user belonging to
noworld has attenuated access privileges. Specifically, when checking such a user's access to
files, the file's permission bits are first ANDed with 0770 for normal files and 0771 for directories.
The effect is to deny world access permissions to noworld users, except when walking into direc-
tories. If the write group exists, then the file system appears read-only to users not in the
group. This is used to make the Plan 9 distribution file server (sources.cs.bell-labs.com) readable
by the world but writable only to the developers.
Fossil starts a new instance of the fossil file server. It is configured mainly through console com-
mands, documented in fossilcons(8).
The options are:
-D    Toggle the debugging flag, which is initially off. When the flag is set, information about

authentication and all protocol messages are written to standard error.
-t    Start a file server console on /dev/cons. If this option is given, fossil does not fork itself

into the background.
-c cmd

Execute the console command cmd. This option may be repeated to give multiple com-
mmands. Typically the only commands given on the command line are “...file,” which exe-
cutes a file containing commands, and “srv −pcons,” which starts a file server console
on /srv/cons. See fossilcons(8) for more information.

-\f file
 Read and execute console commands stored in the Fossil disk file. Conf (q.v.) reads and
writes the command set stored in the disk.

-m Allocate free-memory% percent of the available free RAM for buffers. This overrides all
other memory sizing parameters, notably the -c option to open.

Flchk checks the fossil file system stored in file for inconsistencies. Flchk is deprecated in favor of
the console check command (see fossilcons(8)). Flchk prints fossil console commands that may
be executed to take care of bad pointers (clrp), bad entries (clre), bad directory entries
(clri), unreachable blocks (bfree). Console commands are interspersed with more detailed
commentary on the file system. The commands are distinguished by being prefixed with sharp
signs. Note that all proposed fixes are rather drastic: offending pieces of file system are simply
chopped off.

Flchk does not modify the file system, so it is safe to run concurrently with fossil, though in this
case the list of unreachable blocks and any inconsistencies involving the active file system should
be taken with a grain of salt.

The options are:

-\f Fast mode. By default, flchk checks the entire file system image for consistency, which
includes all the archives to Venti and can take a very long time. In fast mode, flchk avoids
walking in Venti blocks whenever possible.

-c ncache
 Keep a cache of ncache (by default, 1000) file system blocks in memory during the check.

-h host
 Use host as the Venti server.

Flfmt prepares file as a new fossil file system. The file system is initialized with three empty direc-
tories active, archive, and snapshot, as described above. The options are:

-y Yes mode. By default, flfmt will prompt for confirmation before formatting a file that
already contains a fossil file system, and before formatting a file that is not served directly
by a kernel device. If the -y flag is given, no such checks are made.

-b blocksize
 Set the file system block size (by default, 8192).

-h host
 Use host as the Venti server.

-l label
 Set the textual label on the file system to label. The label is only a comment.

-v score
 Initialize the file system using the vac file system stored on Venti at score. The score
should have been generated by fossil rather than by vac(1), so that the appropriate snap-
shot metadata is present.

Conf reads or writes the configuration branded on the Fossil disk file. By default, it reads the con-
figuration from the disk and prints it to standard output. If the -w flag is given, conf reads a new
configuration from config (or else from standard input) and writes it to the disk. Inside the config-
uration file, the argument * may be used to stand in for the name of the disk holding the config-
uration. The Plan 9 kernel boot process runs "fossil -f disk" to start a Fossil file server. The
disk is just a convenient place to store configuration information.

Last prints the vac score that resulted after the most recent archival snapshot of the fossil in file.

EXAMPLES
 Place the root of the archive file system on /n/dump and show the modified times of the MIPS C
compiler over all dumps in December 2002:

  9fs dump
  ls -1 /n/dump/2002/12*/mips/bin/vc
To get only one line of output for each version of the compiler:

```
ls -lp /n/dump/2002/12*/mips/bin/vc | uniq
```

Initialize a new file system, start the server with permission checking turned off, create a users file, and mount the server:

```
fossil/flfmt /dev/sdC0/fossil
fossil/conf −w /dev/sdC0/fossil <<EOF
fsys main config /dev/sdC0/fossil
fsys main open −AWP
fsys main
create /active/adm adm sys d775
create /active/adm/users adm sys 664
users −w
srv −p fscons
srv fossil
EOF
fossil/fossil −f /dev/sdC0/fossil
mount /srv/fossil /n/fossil
```

See the discussion of the users and uname commands in `fossilcons(8)` for more about the user table.

Perhaps because the disk has been corrupted or replaced, format a new file system using the last archive score printed on the console:

```
fossil/flfmt −v b9b3...5559 /dev/sdC0/fossil
```

Note that while /snapshot will be lost, /active and /archive will be restored to their contents at the time of the last archival snapshot.

Blindly accept the changes prescribed by flchk (not recommended):

```
fossil/flchk /dev/sdC0/fossil | sed −n 's/^# //p' >>/srv/fscons
```

A better strategy is to vet the output, filter out any suggestions you’re not comfortable with, and then use the sed command to prepare the script.

**SOURCE**

```
/sys/src/cmd/fossil
```

**SEE ALSO**

`yesterday(1), fs(3), fs(4), srv(4), fossilcons(8), venti(8)`

**BUGS**

It is possible that the disk format (but not the Venti format) will change in the future, to make the disk a full cache rather than just a write buffer. Changing to the new format will require reformatting the disk as in the example above, but note that this will preserve most of the file system (all but /snapshot) with little effort.

The −m option currently assumes a block size of 8K bytes, and a single file system per fossil instance.
NAME
fs - file server, dump

SYNOPSIS
none

DESCRIPTION
The file server was the main file system for Plan 9. It was a stand-alone system that ran on a separate computer. It served the Plan 9 protocol via the IL/IP protocols on Ethernets. The name of the main file server at Murray Hill was emelie.

The file server normally requires all users except none to provide authentication tickets on each attach(5). This can be disabled using the noauth configuration command (see fsconfig(8)).

The group numbered 9999, normally called noworld, is special on the file server. Any user belonging to that group has attenuated access privileges. Specifically, when checking such a user's access to files, the file's permission bits are first ANDed with 0770 for normal files or 0771 for directories. The effect is to deny world access permissions to noworld users, except when walking directories.

The user none is always allowed to attach to emelie without authentication but has minimal permissions.

Emelie maintains three file systems on a combination of disks and write-once-read-many (WORM) magneto-optical disks.

other
is a simple disk-based file system similar to kfs(4).

main
is a worm-based file system with a disk-based look-aside cache. The disk cache holds modified worm blocks to overcome the write-once property of the worm. The cache also holds recently accessed non-modified blocks to speed up the effective access time of the worm. Occasionally (usually daily at 5AM) the modified blocks in the disk cache are dumped. At this time, traffic to the file system is halted and the modified blocks are relabeled to the unwritten portion of the worm. After the dump, the file system traffic is continued and the relabeled blocks are copied to the worm by a background process.

dump
Each time the main file system is dumped, its root is appended to a subdirectory of the dump file system. Since the dump file system is not mirrored with a disk cache, it is read-only. The name of the newly added root is created from the date of the dump: /yyy/mm/dd. Here yyyy is the full year, mm is the month number, dd is the day number and ss is a sequence number if more than one dump is done in a day. For the first dump, ss is null. For the subsequent dumps ss is 1, 2, 3, etc.

The root of the main file system that is frozen on the first dump of March 1, 1992 will be named /1992/0301/ in the dump file system.

EXAMPLES
Place the root of the dump file system on /n/dump and show the modified times of the MIPS C compiler over all dumps in February, 1992:

9fs dump
ls -l /n/dump/1992/02??/mips/bin/vc

To get only one line of output for each version of the compiler:

ls -lp /n/dump/1992/02??/mips/bin/vc | uniq

Make the other file system available in directory /n/emelieother:

mount -c /srv/boot /n/emelieother other

SOURCE
/sys/src/fs

SEE ALSO
yesterday(1), cwf(4), srv(4), fs(8)
Sean Quinlan, “A Cached WORM File System”, Software - Practice and Experience, December,
1991

**BUGS**

For the moment, the file server serves both the old (third edition) and new (fourth edition) versions of 9P, deciding which to serve by sniffing the first packet on each connection.

Required IL, thus now deprecated.
NAME

ftpfs - file transfer protocol (FTP) file system

SYNOPSIS


DESCRIPTION

Ftpfs dials the TCP file transfer protocol (FTP) port, 21, on system and mounts itself (see bind(2)) on mountpoint (default /n/ftp) to provide access via FTP to files on the remote machine. Ftpfs attempts to use FTP’s ‘passive’ mode but falls back to using ‘active’ mode if that fails. If required by the remote machine, ftpfs will ask factotum(4) for a key matching the pattern

proto=pass service=ftp server=system user? !password? keyspec

(If factotum does not have such a key, factotum will prompt the user for one.)

The user names ftp and anonymous conventionally offer guest/read-only access to machines. Anonymous FTP may be called without using factotum by using the –a option and specifying the password.

By default the file seen at the mount point is the user’s remote home directory if he has one. The option –/ forces the mount point to correspond to the remote root. The option –r forces the mount point to correspond to the remote directory remoteroot.

To avoid seeing startup messages from the server use option –q. To see all messages from the server use option –d.

Some systems will hangup an ftp connection that has no activity for a given period. The –K option causes ftp to send a NOP command every 15 seconds to attempt to keep the connection open. This command can cause some servers to hangup, so you’ll have to feel your way.

The –t option causes ftpfs to negotiate TLS encryption with the server.

To terminate the connection, umount (see bind(1)) the mount point.

Since there is no specified format for metadata retrieved in response to an FTP directory request, ftpfs has to apply heuristics to steer the interpretation. Sometimes, though rarely, these heuristics fail. The following options are meant as last resorts to try to steer interpretation.

A major clue to the heuristics is the operating system at the other end. Normally this can be determined automatically using the FTP SYST command. However, in some cases the server doesn’t implement the SYST command. The –o option will force the case by specifying the name of the operating system. Known system types are: UNIX, SUN, TOPS, Plan9, VM, VMS, MVS, NetWare, OS/2, TSO, and WINDOWS_NT.

Some systems and/or FTP servers return directory listings that don’t include the file extension. The –e option allows the user to specify an extension to append to all remote files (other than directories).

Finally, there are two FTP commands to retrieve the contents of a directory, LIST and NLST. LIST is approximately equivalent to ls –l and NLST to ls. Ftpfs normally uses LIST. However, some FTP servers interpret LIST to mean, give a wordy description of the file. Ftpfs normally notices this and switches to using NLST. However, in some rare cases, the user must force the use of NLST with the –n option.

EXAMPLE

You want anonymous FTP access to the system export.lcs.mit.edu. The first import(4) command is only necessary if your machine does not have access to the desired system, but another, called gateway in this example, does.

import gateway /net
ftpfs -a yourname@yourmachine export.lcs.mit.edu

SOURCE

/sys/src/cmd/ip/ftpfs

SEE ALSO

bind(2)
BUGS

Symbolic links on remote Unix systems will always have mode 0777 and a length of 8.

After connecting to a TOPS-20 system, the mount point will contain only one directory, usually /n/ftp/PS:<ANONYMOUS>. However, walking to any valid directory on that machine will succeed and cause that directory entry to appear under the mount point.

Ftpfs caches files and directories. A directory will fall from the cache after 5 quiescent minutes or if the local user changes the directory by writing or removing a file. Otherwise, remote changes to the directory that occur after the directory has been cached might not be immediately visible. Attempting to walk to directory/.flush.ftpfs will flush directory from the cache, thus forcing ftpfs to re-read it.

There is no way to issue the appropriate commands to handle special synthetic FTP file types such as directories that automatically return a `tar` of their contents.

Ftpfs makes copies in /tmp of files being transferred, so its effects might not be immediate. If there is enough main memory, you might want to run ramfs(4) first.

Filenames containing spaces will confuse ftpfs (and other FTP clients).
NAME
  httpfile - serve a single web file

SYNOPSIS
  httpfile [−9d][−c count][−E file][−m mtpt][−s srvname][−x net] url

DESCRIPTION
  Httpfile serves the web page specified by the URL url as a new file file in the directory mtpt. The default file is the last path element of the URL, and the default mtpt is the current directory.

  Httpfile does not download large files all at once. Instead, it requests 64-kilobyte blocks as they are needed to satisfy reads, caching a few blocks in memory at a time.

  The −D and −d options enable a trace of the 9P traffic and general debugging messages.

  The −s option causes httpfile to post the 9P service as /srv/srvname and disables the default mount.

  The −x option specifies an alternate network directory (e.g., /net.alt).

  The −c option sets the number of file blocks kept cached in memory (default 32).

EXAMPLE
  Mount an ISO image on a web server:
    ip/httpfile http://www.9grid.de/plan9/plan9.iso
    9660srv
    mount /srv/9660 /n/iso plan9.iso

SOURCE
/sys/src/cmd/ip/httpfile.c

SEE ALSO
  hget(1), webfs(4)
NAME
ike – IPsec Internet Key Exchange file server

SYNOPSIS
ike [−v][−d what]... [−k key−prefix]

DESCRIPTION
ike implements the IPsec Internet Key Exchange, version 2. It announces on UDP port 500 and represents Security Associations as directories, conventionally in /n/ike.

−v increases verbosity; −d enables debugging output for what. Known values of what include Certdebug, Datadump, Encrdebug, Encrdump, Fsdebug, IKEdebug, Parseddebug, and Verbose. Keys, certificates, etc. are found in files whose name begin with key−prefix; key−prefix defaults to keys/left. See FILES for suffices.

Writing a message of the form ‘newsa saname ip’ to /n/ike/ctl1 will create a Security Association for ip named saname, and create the directory /n/ike/saname, which initially will contain just a ctl file. Removing /n/ike/saname will shut down the Security Association saname.

EXAMPLES
Initiate a Security Association with 135.104.9.152 named bla.

% ike
% mount /srv/ike /n/ike
% cd /n/ike
% echo newsa bla 135.104.9.152 >>ctl
% cat bla/ctl
name: bla
yraddr: 135.104.9.152
role: Initiator
state: resp_state_auth

FILES
/srv/ike posted file server connection
/n/ike usual mount point
/sys/lib/ssl/*.pem PEM certificates
keys/left default key prefix
Cert.cer key suffixes
Cert.pem
KeyClear.asn
KeyClear.pem

SOURCE
/usr/src/cmd/ip/ike

SEE ALSO
rsa(2), ip(3), rsa(8)
/lib/rfc/rfc4306

BUGS
 Doesn’t yet allow creation of child SAs.
 Doesn’t yet get keys from factotum(4).
IKE could be simpler, but apparently IKE v1 was even worse. Negotiates anything that could possibly be negotiated, and uses ASN.1.
NAME
import - import a name space from a remote system

SYNOPSIS
import [ options ] system file [ mountpoint]
import -B [ options ] mountpoint [ cmd [ args ...]]

DESCRIPTION
Import allows an arbitrary file on a remote system to be imported into the local name space. Usually file is a directory, so the complete file tree under the directory is made available.

A process is started on the remote machine, with authority of the user of import, to perform work for the local machine using the exportfs(4) service. The default port used is TCP 17007. If mountpoint is omitted import uses the name of the remote file as the local mount point.

The options are:

- a  -b  -c  -C  Control the construction of union directories, as in mount and bind(1). Only valid when file is a directory.
- A  Skip the authentication protocol. This is useful for connecting to foreign systems like Inferno.
- B  Run in “backwards” mode, described below.
- E  enc  Push an authentication protocol on its network connection. The supported protocols are clear (the default, no protocol) and ssl. There are plans to make tls available.
- e 'enc auth'  Specify the encryption and authentication algorithms to use for encrypting the wire traffic (see ssl(3)). The defaults are rc4_256 and sha1.
- k  keypattern  Use keypattern to select a key to authenticate to the remote side (see auth(2)).
- o  -O  These equivalent flags run import in a pre-9P2000 compatibility mode to import from ancient servers.
- p  Push the aam(8) filter onto the connection to protect against temporary network outages.
- s  name  Post the connection's mountable file descriptor as /srv/ name.

The -B option runs import in "backwards" mode. In this mode, import runs a p9any authentication (as server) over its file descriptor 0 (expected to be an incoming network connection from exportfs -B), mounts the connection onto mntpt, and optionally runs cmd args.

EXAMPLES
Assume a machine kremvax that has IP interfaces for the company intranet and the global internet mounted on /net and /net.alt respectively. Any machine inside the company can get telnet out to the global internet using:

    import -a kremvax /net.alt
telnet /net.alt/tcp!ucbvax

Suppose that the machine moscvax has access to a private file server containing public web pages that need to be served by the less-trusted server webvax. Webvax runs the following listener (see listen(8)) on TCP port 999:

    #!/bin/rc
    import -B -s rowebfs /usr/web /bin/restarthttpd

When moscvax boots, it runs

    exportfs -R -r /usr/web -B tcp!webvax!999

to serve a read-only copy of /usr/web to webvax. When webvax gets the call, import mounts the served tree onto its own /usr/web and then runs /bin/restarthttpd to restart httpd(8).
SOURCE
    /sys/src/cmd/import.c

SEE ALSO
    bind(1), ssl(3), exportfs(4), srv(4), aan(8), listen(8), cs in ndb(8)
NAME
   iostats - file system to measure I/O

SYNOPSIS
   iostats [-d] [-f dbfile] cmd [ args... ]

DESCRIPTION
   iostats is a user-level file server that interposes itself between a program and the regular file server, which allows it to gather statistics of file system use at the level of the Plan 9 file system protocol, 9P. After a program exits a report is printed on standard error.

   The report consists of three sections. The first section reports the amount of user data in read and write messages sent by the program and the average rate at which the data was transferred. The protocol line reports the amount of data sent as message headers, that is, protocol overhead. The rpc line reports the total number of file system transactions.

   The second section gives the number of messages, the fastest, slowest, and average turn around time and the amount of data involved with each 9P message type. The final section gives an I/O summary for each file used by the program in terms of opens, reads and writes.

   If the -d flag is present, a debugging log including all traffic is written to dbfile (default iostats.out).

EXAMPLE
   Display summary of file I/O incurred by ls(1):
      iostats ls
   Start a new shell, displaying all 9P traffic caused by the shell or its children:
      iostats -df /fd/1 rc

SOURCE
   /sys/src/cmd/iostats

SEE ALSO
   dup(3)

BUGS
   Poor clock resolution means that large amounts of I/O must be done to get accurate rate figures.
   Can be fooled by programs that do fresh mounts outside its purview, or by the use of names of files with content that can vary by process (e.g., #d, /dev/cons).
NAME
iokfs - terrorist IP address file system

SYNOPSIS
aux/iokfs [-s service]

DESCRIPTION
Iokfs consults iok to determine if given IP addresses belong to countries approved of by the US
government, and caches the answer. If /mnt/iokok/IP exists, IP is deemed to be from a
non-terrorist country.

The s option posts a mountable file descriptor as /srv/service.

EXAMPLES
See if ehime is in a terrorist country.

% ls /mnt/iokok/135.104.9.17
/mnt/iokok/135.104.9.17

FILES
/mnt/iokok mount point for iokfs

SEE ALSO
iok(8)

BUGS
Only works for IPv4 addresses.
NAME
keyfs, warning – authentication database files

SYNOPSIS
auth/keyfs [-p] [-w [np]] [-mmntpt] [ keyfile ]
auth/warning [-n] [-p]

DESCRIPTION
Keyfs serves a two-level file tree for manipulating authentication information. It runs on the machine providing authentication service for the local Plan 9 network, which may be a dedicated authentication server or a CPU server. The programs described in auth(8) use keyfs as their interface to the authentication database.

Keyfs reads and decrypts file keyfile (default /adm/keys) using the DES key, which is by default read from /nr/NVRAM (see rtc(3)). With option -p, keyfs prompts for a password from which the key is derived. Keyfile holds a 41-byte record for each user in the database. Each record is encrypted separately and contains the user's name, DES key, status, host status, and expiration date. The name is a null-terminated UTF string NAMELEN bytes long. The status is a byte containing binary 0 if the account is enabled, 1 if it is disabled. Host status is a byte containing binary 1 if the user is a host, 0 otherwise. The expiration date is four-byte little-endian integer which represents the time in seconds since the epoch (see date(1)) at which the account will expire. If any changes are made to the database that affect the information stored in keyfile, a new version of the file is written.

There are two authentication databases, one for Plan 9 user information, and one for SecureNet user information. A user need not be installed in both databases but must be installed in the Plan 9 database to connect to a Plan 9 server.

Keyfs serves an interpretation of the keyfile in the file tree rooted at mntpt (default /mnt/key). Each user user in keyfile is represented as the directory mntpt/user.

Making a new directory in mntpt creates a new user entry in the database. Removing a directory removes the user entry, and renaming it changes the name in the entry. Such changes are reflected immediately in keyfile. Keyfs does not allow duplicate names when creating or renaming user entries.

All files in the user directories except for key contain UTF strings with a trailing newline when read, and should be written as UTF strings with or without a trailing newline. Key contains the DESKEYLEN-byte encryption key for the user.

The following files appear in the user directories.

key The authentication key for the user. If the user's account is disabled or expired, reading this file returns an error. Writing key changes the key in the database.

log The number of consecutive failed authentication attempts for the user. Writing the string bad increments this number; writing good resets it to 0. This number is not stored in keyfile, and is initialized to 0 when keyfs starts. When the number reaches a multiple of ten, keyfs temporarily disables the account for that many seconds. Reads from the key or secret files during this time return the error "user in purgatory."

status The current status of the account, either ok or disabled. Writing ok enables the account; writing disabled disables it.

expire The expiration time for the account. When read, it contains either the string never or the time in seconds since the epoch that the account will expire. When written with strings of the same form, it sets the expiration date for the user. If the expiration date is reached, the account is not disabled, but key cannot be read without an error.

If the -w option is on, keyfs runs the command warning once every 24 hours to mail people about expiring keys. Warnings are sent 14 days and 7 days prior to expiration. The argument to -w, either p or n, is passed to warning to restrict the warnings to the Plan 9 or SecureNet database.

The default for keyfs is not to call warning at all; warning's own default is to warn about both. The files /adm/netkeys.who and /adm/keys.who are used to find the mail addresses to send to. The first word on each line identifies a user. Any subsequent strings on the line delimited '<' and '>' are considered mail addresses to send warnings to. If multiple lines match a user, the last in the file is used. Changeuser (see auth(8)) adds lines to these files.
FILES

/adm/keys  Encrypted key file for the Plan 9 database.
/adm/netkeys  Encrypted key file for the SecureNet database.
/adm/keys.who  List of users in the Plan 9 database.
/adm/netkeys.who  List of users in the SecureNet database.
#r/nvram  The non-volatile RAM on the server, which holds the key used to decrypt key files.

SOURCE
/sys/src/cmd/auth/keyfs.c
/sys/src/cmd/auth/warning.c

SEE ALSO
authsrv(6), namespace(6), auth(8)
NAME
kfs – disk file system

SYNOPSIS
disk/kfs [ -rc ][ -b n ][ -f file ][ -n name ][ -p perm ][ -s ][ -B nbuf ]

DESCRIPTION
Kfs is an old, local user-level file server for a Plan 9 terminal with a disk. It maintains a hierarchical Plan 9 file system on the disk and offers 9P (see intro(5)) access to it. Kfs begins by checking the file system for consistency, rebuilding the free list, and placing a file descriptor in /srv/name, where name is the service name (default kfs). If the file system is inconsistent, the user is asked for permission to ream (q.v.) the disk. The file system is not checked if it is reamed.

The options are

b n  If the file system is reamed, use n byte blocks. Larger blocks make the file system faster and less space efficient. 1024 and 4096 are good choices. N must be a multiple of 512.

f file Use file as the disk. The default is /dev/sdC0/fs.

n name Use kfs.name as the name of the service.

p perm Use perm as the initial permissions for the command channel /srv/service.cmd; the default is 660.

r Ream the file system, erasing all of the old data and adding all blocks to the free list.

s Post file descriptor zero in /srv/service and read and write protocol messages on file descriptor one.

B Allocate nbuf in-memory file system blocks. The default is as many as will fit in 10% of memory or two megabytes, whichever is smaller.

EXAMPLES
Create a file system with service name kfs.local and mount it on /n/kfs.

% disk/kfs -rb4096 -nlocal
% mount -c /srv/kfs.local /n/kfs

FILES
/dev/sdC0/fs  Default file holding blocks.

SOURCE
/sys/src/cmd/disk/kfs

SEE ALSO
fossil(4), kfs.cmd(8), mkfs(8), prep(8), sd(3)

BUGS
For the moment, kfs serves both the old (third edition) and new (fourth edition) versions of 9P, deciding which to serve by sniffing the first packet on each connection.

Kfs doesn’t allow creating files with component names longer than 28 bytes.
NAME

Lnfs – long name file system

SYNOPSIS

```
lnfs [−r][−s srvname] mountpoint
unlnfs mountpoint
```

DESCRIPTION

*Lnfs* starts a process that mounts itself (see *bind(2)*) on *mountpoint*. It presents a filtered view of the files under the mount point, allowing users to use long file names on file servers that do not support file names longer than 27 bytes.

The names used in the underlying file system are the base32 encoding of the md5 hash of the longer file name. The user need not know the mapping since *Lnfs* does all the work. *Lnfs* maintains a file `.longnames` in the directory *mountpoint* to record the long file names.

The options are:

- `−r` allow only read access to the file system
- `−s` provide a service name, `srvname`, to post in `/srvv`. Without this option, no posting is performed.

*Unlnfs* renames files with shortened names to their actual long names. It is useful once you have moved to a file server with true long name support.

FILES

`.longnames`

SOURCE

`/sys/src/cmd/lnfs.c`
`/sys/src/cmd/unlnfs.c`

BUGS

This exists only to shame us into getting a real long name file server working.
NAME
mntgen - automatically generate mount points for file systems

SYNOPSIS
mntgen[−s service][ mnt]

DESCRIPTION
Mntgen mounts itself on mnt (default /n) after the current contents, creating subdirectories on demand as they are accessed. It is intended to supply mount points automatically.

The −s option causes mntgen to post a 9P service file in /srv/service.

SOURCE
/sys/src/cmd/mntgen.c
NAME
namespace – structure of conventional file name space

SYNOPSIS
none

DESCRIPTION
After a user's profile has run, the file name space should adhere to a number of conventions if the
system is to behave normally. This manual page documents those conventions by traversing the
file hierarchy and describing the points of interest. It also serves as a guide to where things reside
in the file system proper. The traversal is far from exhaustive.

First, here is the appearance of the file server as it appears before any mounts or bindings.

/ The root directory.
/adm The administration directory for the file server.
/adm/users List of users known to the file server; see users(6).
/adm/keys Authentication keys for users.
/adm/netkeys SecureNet keys for users; see securenet(8).
/adm/timezone Directory of timezone files; see ctime(2).
/adm/timezone/EST.EDT
Time zone description for Eastern Time. Other such files are in this directory
too.
/adm/timezone/timezone
Time zone description for the local time zone; a copy of one of the other files
in this directory.

/bin
/dev
/env
/fd
/net
/proc
/srv
/tmp
All empty unwritable directories, place holders for mounted services and
directories.
/mnt
A directory containing mount points for applications.
/n
A directory containing mount points for file trees imported from remote sys-
tems.

/386
/68000
/68020
/alpha
/arm
/mips
/power
/sparc
Each CPU architecture supported by Plan 9 has a directory in the root contain-
ing architecture-specific files, to be selected according to $objtype or
$cputype (see 2c(1) and init(8)). Here we list only those for /386.

/386/init
The initialization program used during bootstrapping; see init(8).
/386/bin
Directory containing binaries for the Intel x86 architecture.
/386/bin/aux
/386/bin/ip
/etc.
Subdirectories of /386/bin containing auxiliary tools and collecting related
programs.
/386/lib
Directory of object code libraries as used by 81 (see 2l(1)).
/386/include
Directory of x86–specific C include files.
/386/9*
The files in /386 beginning with a 9 are binaries of the operating system or
its bootstrap loader.
/386/mkfile
Selected by mk(1) when $objtype is 386, this file configures mk to compile
for the Intel x86 architecture.
/rc Isomorphic to the architecture-dependent directories, this holds executables and libraries for the shell, rc(1).
/rc/bin Directory of shell executable files.
/rc/lib Directory of shell libraries.
/rc/lib/rcmain Startup code for rc(1).
/lib Collections of data, generally not parts of programs.
/lib/mammals Databases.
/lib/sky The network database used by the networking software; see ndb(6) and ndb(8).
/lib/namespace The file used by newns (see auth(2)) to establish the default name space; see namespace(6).
/lib/font/bit Bitmap font files.
/lib/font/hershey Vector font files.
/lib/rfc Directory of Internet 'Requests For Comments', ranging from trivia to specifications.
/lib/rfc/grabrfc Maintains RFC collection; usually run from cron (see auth(8)).
/sys System software.
/sys/include Directory of machine-independent C include files.
/sys/lib Pieces of programs not easily held in the various bins.
/sys/lib/acid Directory of acid(1) load modules.
/sys/lib/dist Software used to assemble the distribution's installation floppy.
/sys/lib/troff Directory of troff(1) font tables and macros.
/sys/lib/yaccpar The yacc(1) parser.
/sys/man The manual.
/sys/doc Other system documentation.
/sys/log Log files created by various system services.
/sys/src Top-level directory of system sources.
/sys/src/cmd Source files created in the bin directories.
/sys/src/9 Source to the operating system for terminals and CPU servers.
/sys/src/fs Source to the operating system for file servers.
/sys/src/lib* Source to the libraries.
/usr A directory containing home directories of users.
/mail Directory of electronic mail; see mail(1).
/mail/box Directory of users' mail box files.
/mail/lib Directory of alias files, etc.
/acme Directory of tools for acme(1).
/cron Directory of files for cron(8).
/cfg/system System-specific files, often addenda to their namesakes, notably cpurc, termsc, namespace, and consoledb.

The following files and directories are modified in the standard name space, as defined by /lib/namespace (see namespace(6)).

/
The root of the name space. It is a kernel device, root(3), serving a number of local mount points such as /bin and /dev as well as the bootstrap program /boot. Unioned with / is the root of the main file server.
/boot Compiled into the operating system kernel, this file establishes the connection to the main file server and starts init; see boot(8) and init(8).
/bin Mounted here is a union directory composed of /$objtype/bin, /rc/bin, $home/$objtype/bin, etc., so /bin is always the directory containing the appropriate executables for the current architecture.
/dev Mounted here is a union directory containing I/O devices such as the console (cons(3)), the interface to the raster display (draw(3)), etc. The window
system, rio(1), prefixes this directory with its own version, overriding many device files with its own, multiplexed simulations of them.

/env
Mounted here is the environment device, env(3), which holds environment variables such as $cputype.

/net
Mounted here is a union directory formed of all the network devices available.

/net/cs
The communications point for the connection server, ndb/cs (see ndb(8)).

/net/dns
The communications point for the Domain Name Server, ndb/dns (see ndb(8)).

/net/tcp
Directories holding the IP protocol devices (see ip(3)).

/net/udp
Mounted here is the process device, proc(3), which provides debugging access to active processes.

/fd
Mounted here is the dup device, dup(3), which holds pseudonyms for open file descriptors.

/srv
Mounted here is the service registry, srv(3), which holds connections to file servers.

/srv/boot
The communication channel to the main file server for the machine.

/srv/boot
Mount point for factotum(4).

/mnt/wsys
Mount point for the window system.

/mnt/term
Mount point for the terminal's name space as seen by the CPU server after a cpu(1) command.

/n/kremvax
A place where machine kremvax's name space may be mounted.

/tmp
Mounted here is each user's private tmp, $home/tmp.

SEE ALSO
intro(1), namespace(6)
NAME
nfs – Sun network file system client

SYNOPSIS
aux/portmap [-R] host cmd
aux/nfsmount [-R] host cmd

DESCRIPTION
Nfs translates between the Sun network file system protocol (NFS) and 9P, allowing 9P clients to mount file systems on NFS servers. NFS servers comprise two separate services: a mount service used to obtain the initial file handle, and a file service used to perform actual file system operations. The Sun port mapper service is typically used to find these two services. If one address is given, it is taken to be the address of a port mapper service; nfs queries the port mapper to find the addresses of the NFS mount service and file service. If two addresses are given, the port mapper is bypassed; addr1 is used as the address of the NFS mount service, and addr2 is used as the address of the file service.

The options are:
-D print all 9P messages.
-R print all NFS messages.
-v print verbose information about session startup.
-p perm
set the posted service file to have mode perm, which is assumed to be octal; the default is 600.
-s srvname
post the service as /srv/srvname; the default is /srv/addr1.
-u passwd group
translate user and group names using the passwd and group files, which are in the traditional Unix format. The translation is used to present names for user and group in stat(5) and wstat messages. The translation is also used to choose the user and group credentials to present for a user. Without this option, users and groups are presented as decimal numbers, and everyone attaches as uid -1 (nobody on most Unix systems).

Portmap and nfsmount are test programs to perform port mapper and NFS mount RPCs. They are useful mainly to help debug problems with starting nfs itself. The –R option causes them to print all RPC messages sent and received.

Portmap queries a Sun RPC portmap server, which maps integer (program, version, protocol) triples to port numbers. Program and version are Sun RPC defined, while protocol is typically TCP (6) or UDP (17). The commands are:
null a no-op
dump print the entire map
set prog vers proto port
add an entry to (or replace an entry in) the map
unset prog vers proto port
remove an entry from the map
getport prog vers proto
look for an entry with prog, vers, proto in the map, and return the corresponding port The default command is dump. For running NFS over UDP, there must be an entry for the NFS v3 mount daemon (100003, 3, 17) and the NFS v3 server itself (100003, 3, 17).

Nfsmount queries a Sun NFS mount server, which authenticates (ha!) connections and hands out file handles naming the root of an exported file system. This handle is used as the basis for a conversation with the NFS service daemon itself. The commands are:
null  a no-op

export
dump the export table; each line is a path followed by a list of machines or groups allowed
to mount that path

mnt  path
attempt to acquire a file handle for path. the request has user and group id 1001 and
not as the system name.

umnt  path
notify the mount daemon that a particular path is being unmounted by the requesting sys-

umntall
notify the mount daemon that all paths mounted by the requesting system are being

dump  should also dump an export table, but typically does nothing

EXAMPLE
We use this in our /rc/bin/9fs script to mount all the home directories served by bopp:

```bash
case bopp
  if(! test -f /srv/bopp)
    nfs -p 666 -u /lib/ndb/1127.passwd /lib/ndb/1127.group bopp
  umount /n/bopp >[2]/dev/null
  for(i in u0 u1 u2 u3 u4 u5 u6 u7 u8 u9)
    mount -a /srv/bopp /n/bopp /$i
```

SOURCE
/sys/src/cmd/nfs.c
/sys/src/libsunrpc

SEE ALSO
nfsserver(8), srv(4)

BUGS
The authentication employed by NFS is laughable. The server simply trusts the uid, gid, and group
list presented by the client.

Nfs speaks only NFS version 3. Older operating systems typically have reasonable NFS version 2
servers but crash when serving version 3.
NAME
nntpfs - network news transport protocol (NNTP) file system

SYNOPSIS
nntpfs [-a] [-s service] [-m mountpoint] [system]

DESCRIPTION
Nntpfs dials the TCP network news transport protocol (NNTP) port, 119, on system (default "$nntp") and presents at mountpoint (default /mnt/news) a file system corresponding to the news articles stored on system.

If the -s option is given, the file system is posted as /srv/service. If the -a option is given, nntpfs authenticates to the system with a user name and password obtained from factotum(4). The key specifier is

proto=pass service=nntp server=server? !password?

The file system contains a directory per newsgroup, with dots turned into slashes, e.g., comp/os/plan9 for comp.os.plan9. Each newsgroup directory contains one numbered directory per article. The directories follow the numbering used by the server. Each article directory contains three files: article, header, and body. The article file contains the full text of the article, while header and body contain only the header or body.

Each newsgroup directory contains a write-only post file that may be used to post news articles. RFC1036-compliant articles should be written to it. The post file will only exist in a given newsgroup directory if articles are allowed to be posted to it. Other than that, the post file is not tied to its directory's newsgroup. The groups to which articles are eventually posted are determined by the newsgroups: header lines in the posted article, not by the location of the post file in the file system.

The qid version of a newsgroup directory is the largest numbered article directory it contains (~0, if there are no articles).

The modification time on a newsgroup directory is the last time a new article was recorded during this nntpfs session. To force a check for new articles, stat(2) the newsgroup directory.

To force a check for new newsgroups, stat(2) the root directory. Note that this causes the entire list of groups, which can be about a megabyte, to be transferred.

To terminate the connection, umount the mount point.

Nntpfs makes no effort to send "keepalives" so that servers do not hang up on it. Instead, it redial as necessary when hangups are detected.

EXAMPLE
Authenticate to a private news server:

% echo key proto=pass service=nntp server=nose.mit.edu \
user=rsc !password=secret >/mnt/factotum/ctl
% nntpfs -a nose.mit.edu

SOURCE
/sys/src/cmd/nntpfs.c

BUGS
Directories are presented for deleted articles; the files in them cannot be opened.
NAME

paqfs - compressed read-only file system

SYNOPSIS


DESCRIPTION

Paqfs interprets the compressed read-only file system created by mkpaqfs(8) and stored in paqfile so that it can be mounted into a Plan 9 file system. Paqfs is typically used to create a stand alone file system for a small persistent storage device, such as a flash ROM. It does not authenticate its clients and assumes each group has a single member with the same name.

Options to paqfs are:

- `c cachesize`
  The number of file system blocks to cache in memory. The default is 20 blocks.

- `M mesgsize`
  The maximum 9P message size. The default is sufficient for 8K byte read message.

- `d`
  Output various debugging information to stderr.

- `i`
  Use file descriptors 0 and 1 as the 9P communication channel rather than create a pipe.

- `q`
  Suppress the output of the archive creation date and fingerprint to stderr.

- `m mtpt`
  The location to mount the file system. The default is /n/paq.

- `s`
  Post the 9P channel on `#s/srvname`, default `#s/paqfs`, rather than mounting it on mtpt.

- `S`
  The name to post in `#s`. The default is paqfs.

- `p`
  Both post the 9P channel in `#s` and mount the paqfile in to the filesystem.

- `v`
  Verify the integrity of the paqfile. Before mounting the file system, the entire file is parsed and the sha1 checksum of the file system data is compared to the checksum embedded in the file. This option enables the use of paqfs with files that consist of a paq file system concatenated with additional data.

SOURCE

/sys/src/cmd/paqfs/paqfs.c

SEE ALSO

mkpaqfs(8)
NAME
plumber - file system for interprocess messaging

SYNOPSIS
plumber [-p plumbing]

DESCRIPTION
The plumber is a user-level file server that receives, examines, rewrites, and dispatches plumb(6)
messages between programs. Its behavior is programmed by a plumbing file (default
/usr/$user/lib/plumbing) in the format of plumb(6).

Its services are mounted on the directory /mnt/plumb (/mnt/term/mnt/plumb on the CPU
server) and consist of two pre-defined files, send and rules, and a set of output ports for dis-
patching messages to applications. The service is also published as a srv(4) file, named in
$plumbsrv, for mounting elsewhere.

Programs use write (see read(2)) to deliver messages to the send file, and read(2) to receive
them from the corresponding port. For example, sam(l)'s plumb menu item or the B command
cause a message to be sent to /mnt/plumb/send; sam in turn reads from, by convention,
/mnt/plumb/edit to receive messages about files to open.

A copy of each message is sent to each client that has the corresponding port open. If none has it
open, and the rule has a plumb client or plumb start rule, that rule is applied. A plumb
client rule causes the specified command to be run and the message to be held for delivery
when the port is opened. A plumb start rule runs the command but discards the message. If
neither start or client is specified and the port is not open, the message is discarded and a
write error is returned to the sender.

The set of output ports is determined dynamically by the specification in the plumbing rules file: a
port is created for each unique destination of a plumb to rule.

The set of rules currently active may be examined by reading the file /mnt/plumb/rules; appending to this file adds new rules to the set, while creating it (opening it with OTRUNC) clears
the rule set. Thus the rule set may be edited dynamically with a traditional text editor. However,
ports are never deleted dynamically; if a new set of rules does not include a port that was defined
in earlier rules, that port will still exist (although no new messages will be delivered there).

FILES
/usr/$user/lib/plumbing default rules file
/sys/lib/plumb directory to search for files in include statements
/mnt/plumb mount point for plumber(4).

SOURCE
/sys/src/cmd/plumb

SEE ALSO
plumb(1), plumb(2), plumb(6)

BUGS
Plumber's file name space is fixed, so it is difficult to plumb messages that involve files in newly
mounted services.
NAME
ramfs - memory file system

SYNOPSIS
ramfs [-Dipsu] [-m mountpoint] [-S srvname]

DESCRIPTION
Ramfs starts a process that mounts itself (see bind(2)) on mountpoint (default /tmp). The ramfs process implements a file tree rooted at dir, keeping all files in memory. Initially the file tree is empty.

The −D option enables a trace of general debugging messages.

The −i flag tells ramfs to use file descriptors 0 and 1 for its communication channel rather than create a pipe. This makes it possible to use ramfs as a file server on a remote machine: the file descriptors 0 and 1 will be the network channel from ramfs to the client machine.

The −p flag causes ramfs to make its memory 'private' (see proc(3)) so that its files are not accessible through the debugging interface.

The −s (−S) flag causes ramfs to post its channel on /srv/ramfs (/srv/srvname) rather than mounting it on mountpoint, enabling multiple clients to access its files. However, it does not authenticate its clients and its implementation of groups is simplistic, so it should not be used for precious data.

The −u option permits ramfs to consume as much memory as needed; without it, ramfs will limit its consumption to some arbitrary amount, currently 768MB (enough to hold a CD image).

This program is useful mainly as an example of how to write a user-level file server. It can also be used to provide high-performance temporary files.

SOURCE
/sys/src/cmd/ramfs.c

SEE ALSO
bind(2)
NAME
ratfs - mail address ratification file system

SYNOPSIS
ratfs [-d] [-c configuration] [-f classification] [-m mountpoint]

DESCRIPTION
Ratfs starts a process that mounts itself (see bind(2)) on mountpoint (default /mail/ratify).
Ratfs is a persistent representation of the local network configuration and spam blocking list.
Without it each instance of smtpd(6) would need to reread and parse a multigigabyte list of
addresses and accounts.
Ratfs serves a control file, ctl, and several top level directories: trusted, deny, dial, block, delay, and allow.
The control file is write only and accepts three possible commands:
reload reads classification and configuration
debug file creates file and sends debugging output to it.
nodebug closes the debug file and turns off debugging

The directory trusted serves a file for each IP range from which all mail is trusted. The names
of the files are CIDR blocks; an IP address or an IP address followed by #n, where n is the number
of bits to match. To check if any IP address falls in a trusted range, it is sufficient to open the file
whose name is the IP address. For example, if trusted contains only the file
135.104.0.0#16, an attempt to open the file 135.104.9.1 will succeed while opening 10.1.1.1
will fail. To determine the particular range matched, dirfstat (see stat(2)) the open file and
the name field will be the matching CIDR range.

The trusted ranges come both from the ournet entries in the file configuration (default
/mail/lib/block(2)) and from creates, typically done by imap4d (see ipserv(8)) and pop3
(see mail(1)) whenever they are used to read someone’s mail.

The remaining directories, allow, block, delay, deny, and dial, represent the contents of the
classification (default /mail/lib/smtpd.conf.ext). Each contains two directories; ip and
account. The ip directory has the same open semantics as the trusted directory, i.e., to
check if an IP address falls in that category, try to open a file whose name is the IP address. The
account directory is similar but is used for matching strings. Each file in the directory repre-
sents a regular expression. To see if one of the strings matches one of the regular expressions,
try to open the file whose name is the string. If it succeeds, then there is a regular expression that
matches. To determine the regular expression, fstat the open file. The name field will be the
regular expression.

There is a direct mapping from entries in classification and files under allow, block, delay,
deny, and dial. A configuration file entry of the form:
dial 135.104.9.0/24
 corresponds to the file dial/ip/135.104.9.0#24. An entry of the form
 *block .*!gre
 corresponds to the file block/account/.!*gre.

Both the configuration file and control file formats are described in smtpd(6).

SOURCE
/sys/src/cmd/ratfs

SEE ALSO
mail(1) smtpd(6) scanmail(8)
NAME
   rdbfs - remote kernel debugging file system

SYNOPSIS
   rdbfs [-d][-p pid][-t text][ device ]

DESCRIPTION
   Rdbfs presents in /proc/pid (default /proc/1) a set of process files for debugging a kernel
   over the serial line device (default /dev/eia0).

   The text file presented is just a copy of text (default /386/9pc). It can usually be ignored,
   since the debuggers open kernel files directly rather than using /proc/n/text.

   Kernels can be remotely debugged only when they are suspended and serving a textual debugging
   protocol over their serial lines. Typing "^t^td" (control-t, control-t, d) at the console will
   cause the kernel to enter this mode when it 'panics'. Typing "^t^tD" causes the kernel to enter
   this mode immediately.

   Because the debugging protocol is textual, a console provided by consolefs(4) may be substituted
   for the serial device.

SOURCE
   /sys/src/cmd/rdbfs.c
   /sys/src/9/port/rdb.c

SEE ALSO
   acid(1), db(1), consolefs(4)
NAME
rio - window system files

SYNOPSIS
rio [-i 'cmd'] [-s] [-f font]

DESCRIPTION
The window system rio serves a variety of files for reading, writing, and controlling windows. Some of them are virtual versions of system files for dealing with the display, keyboard, and mouse; others control operations of the window system itself. Rio posts its service in the /srv directory, using a name constructed from a catenation of the user ID and a process id; the environment variable $wsys is set to this service name within processes running under the control of each invocation of rio. Similarly, rio posts a named pipe to access the window creation features (see window in rio(1)) from outside its name space; this is named in $wctl.

A mount (see bind(1)) of $wsys causes rio to create a new window; the attach specifier in the mount gives the coordinates of the created window. The syntax of the specifier is the same as the arguments to window (see rio(1)). By default, the window is sized and placed automatically. It is always necessary, however, to provide the process id of the process to whom to deliver notes generated by DEL characters and hangups in that window. That pid is specified by including the string -pid pid in the attach specifier. (See the Examples section q.v.)

When a window is created either by the window command (see rio(1)) or by using the menu supplied by rio, this server is mounted on /mnt/wsys and also /dev; the files mentioned here appear in both those directories.

Some of these files supply virtual versions of services available from the underlying environment, in particular the character terminal files cons(3), and the mouse files mouse(3) and cursor, each specific to the window. Note that the draw(3) device multiplexes itself; rio places windows but does not mediate programs' access to the display device.

Other files are unique to rio.

cons is a virtual version of the standard terminal file cons(3). Rio supplies extra editing features and a scroll bar (see rio(1)).

conscctl controls interpretation of keyboard input. Writing strings to it sets these modes: rawon turns on raw mode; rawoff turns off raw mode; holdon turns on hold mode; holdoff turns off hold mode. Closing the file makes the window revert to default state (raw off, hold off).

cursor Like mouse (q.v.), a multiplexed version of the underlying device file, in this case representing the appearance of the mouse cursor when the mouse is within the corresponding window.

label initially contains a string with the process ID of the lead process in the window and the command being executed there. It may be written and is used as a tag when the window is hidden.

mouse is a virtual version of the standard mouse file (see mouse(3)). Opening it turns off scrolling, editing, and rio--supplied menus in the associated window. In a standard mouse message, the first character is m, but rio will send an otherwise normal message with the first character x if the corresponding window has been resized. The application must then call getwindow (see graphics(2)) to re-establish its state in the newly moved or changed window. Reading the mouse file blocks until the mouse moves or a button changes. Mouse movements or button changes are invisible when the mouse cursor is located outside the window, except that if the mouse leaves the window while a button is pressed, it will continue receiving mouse data until the button is released.

screen is a read-only file reporting the depth, coordinates, and raster image corresponding to the entire underlying display, in the uncompressed format defined in image(6).

snarf returns the string currently in the snarf buffer. Writing this file sets the contents of the snarf buffer. When rio is run recursively, the inner instance uses the snarf buffer of the parent, rather than managing its own.

text returns the full contents of the window. It may not be written.

wctl may be read or written. When read, it returns the location of the window as four decimal integers formatted in the usual 12–character style: upper left x and y, lower right x
and y. Following these numbers are strings describing the window's state: hidden or visible; current or not current. A subsequent read will block until the window changes size, location, or state. When written to, wctl accepts messages to change the size or placement of the associated window, and to create new windows. The messages are in a command–line like format, with a command name, possibly followed by options introduced by a minus sign. The options must be separated by blanks, for example –dx 100 rather than –dx100.

The commands are resize (change the size and position of the window), move (move the window), scroll (enable scrolling in the window), noscroll (disable scrolling), set (change selected properties of the window), top (move the window to the 'top', making it fully visible), bottom (move the window to the 'bottom', perhaps partially or totally obscuring it), hide (hide the window), unhide (restore a hidden window), current (make the window the recipient of keyboard and mouse input), and new (make a new window). The top and bottom commands do not change whether the window is current or not; the others always make the affected window current.

Neither top nor bottom has any options. The resize, move, and new commands accept –minx n, –miny n, –maxx n, and –maxy n options to set the position of the corresponding edge of the window. They also accept an option –r miny maxx to set all four at once. The resize and new commands accept –dx n and –dy n to set the width and height of the window. By default, rio will choose a convenient geometry automatically.

Finally, the new command accepts an optional shell command and argument string, given as plain strings after any standard options, to run in the window instead of the default rc –i (see rc(1)). The –pid pid option to new identifies the pid of the process whose 'note group' should receive interrupt and hangup notes generated in the window. The initial working directory of the new window may be set by a –cd directory option. The –hide option causes the window to be created off-screen, in the hidden state, while –scroll and –noscroll set the initial scrolling state of the window; the default is that of the main program.

The set command accepts a set of parameters in the same style; only –pid pid is implemented.

So programs outside name spaces controlled by rio may create windows, wctl new messages may also be written to the named pipe identified by $wctl.

wdir is a read/write text file containing rio's idea of the current working directory of the process running in the window. It is used to fill in the wdir field of plumb(6) messages rio generates from the plumb menu item on button 2. The file is writable so the program may update it; rio is otherwise unaware of chdir(2) calls its clients make. In particular, rc(1) maintains /dev/wdir in default rio(1) windows.

winid returns the unique and unchangeable ID for the window; it is a string of digits.

window is the virtual version of /dev/screen. It contains the depth, coordinates, and uncompressed raster image corresponding to the associated window.

wsys is a directory containing a subdirectory for each window, named by the unique ID for that window. Within each subdirectory are entries corresponding to several of the special files associated with that window: cons, consctl, label, mouse, etc.

EXAMPLES

Cause a window to be created in the upper left corner, and the word hi to be printed there.

    mount $wsys /tmp 'new −r 0 0 128 64 −pid ' $pid
    echo hi > /tmp/cons

Start sam(1) in a large horizontal window.

    echo new −dx 800 −dy 200 −cd /sys/src/cmd sam > /dev/wctl

Print the screen image of window with id 123.

    lp /dev/wsystab/123/window

SOURCE

/sys/src/cmd/rio
SEE ALSO

rio(1), draw(3), mouse(3), cons(3), event(2), graphics(2).
NAME
sacfs - compressed file system

SYNOPSIS
disk/sacfs [−i infd outfd] [−s] [−m mountpoint] file

DESCRIPTION
Sacfs interprets the compressed, block based file system created by mksacfs(8) and stored in file so that it can be mounted into a Plan 9 file system. Sacfs is typically used to create a stand alone file system from a small persistent storage device, such as a flash rom. It does not authenticate its clients and assumes each group has a single member with the same name.

The −s flag causes sacfs to post its channel on #s/sacfs. The −i flag causes sacfs to use file descriptors infd and outfd for its communication channel. If neither −s nor −i are given, sacfs mounts itself on mountpoint (default /n/c:).

SOURCE
/sys/src/cmd/disk/sacfs/sacfs.c

SEE ALSO
mksacfs(8)
NAME
snap, snapfs - create and mount process snapshots

SYNOPSIS
snap [-o file] pid...
snapfs [-a] [-m mtpt] [-s service] file...

DESCRIPTION
Snap and snapfs allow one to save and restore (static) process images, usually for debugging on a
different machine or at a different time.
Snap writes a snapshot (see snap(6)) of the named processes to file (default standard output). If
pid is a text string rather than a process id, snap will save all processes with that name that are
owned by the current user. Both memory and text images are saved.
Snapfs is a file server that recreates the /proc directories for the processes in the snapshot. By
default, it mounts the new directories into /proc before the current entries. The -m option can
be used to specify an alternate mountpoint, while -a will cause it to mount the new directories
after the current entries. The -s option causes it to serve requests via /srv/service.

EXAMPLE
Suppose page has hung viewing Postscript on your terminal, but the author is gone for the rest of
the month and you want to make sure the process is still around for debugging on his return. You
can save the errant processes with
    snap -o page.snap '{psu | awk '$NF ~ /page|gs/ {print $2}'}
When the author returns, he can add the process images to his name space by running
    snapfs page.snap
and then use a conventional debugger to debug them.

SOURCE
/sys/src/cmd/snap

SEE ALSO
    acid(1), db(1), proc(3), snap(6)

BUGS
The snapshots take up about as much disk space as the processes they contain did memory. Com-
pressing them when not in use is recommended, as is storing them on a rewritable disk.
Pid as a non–numeric string is unimplemented; it has to be a number.
NAME
srv, svr Old 9 p, 9fs, srv ssh - start network file service

SYNOPSIS
srv [-abcCemnq] [-s seconds] [net!] system! service [ srvname [ mtpt ]]
srv ssh [-r] [-R] [-s] [-u u9fs path] system [ srvname [ mtpt ]]
9fs [net!] system [mountpoint]
srv Old 9p [-abcCdF] [-p servicename] [-s] [-m mountpoint] [-u user] [-x command]
     [-n network−addr | -f file]

DESCRIPTION
Srv dials the given machine and initializes the connection to serve the 9P protocol. By default, it connects to the 9fs (9P) service, which for TCP is port 564. It then creates in /srv a file named srvname. Users can then mount (see bind(1)) the service, typically on a name in /n, to access the files provided by the remote machine. If srvname is omitted, the first argument to srv is used.
Option m directs srv to mount the service on /n/system or onto mtpt if it is given. Option q suppresses complaints if the /srv file already exists. The a, b, c, C, and n options are used to control the mount flags as in mount (see bind(1)). The e option causes srv to treat system as a shell command to be executed rather than an address to be dialed. The s option causes srv to sleep for the specified number of seconds after establishing the connection before posting and mounting it. This is sometimes needed by srv ssh.
The specified service must serve 9P. Usually service can be omitted; when calling some non−Plan−9 systems, a service such as u9fs must be mentioned explicitly.
The 9fs command does the srv and the mount necessary to make available the files of system on network net. The files are mounted on mountpoint, if given; otherwise they are mounted on /n/system. If system contains / characters, only the last element of system is used in the /n name.
9fs recognizes some special names, such as dump to make the dump file system available on /n/dump. 9fs is an rc(1) script; examine it to see what local conventions apply.
Srv ssh is an rc(1) command that connects to a remote Unix system via ssh(1) and starts u9fs(4). The −u option specifies the path to the u9fs binary on the remote system. (By default, an unrooted path of u9fs is used; if the binary is in the path of the remote SSH server, you don’t need the −u option.) For information about the other options, see the introductory comment in /rc/bin/srv ssh. The arguments are the same as srv.
Srv Old 9p is a compatibility hack to allow Fourth Edition Plan 9 systems to connect to older 9P servers. It functions as a variant of srv that performs a version translation on the 9P messages on the underlying connection. Some of its options are the same as those of srv; the special ones are:

−d Enable debugging.
−F Insert a special (internal) filter process to the connection to maintain message boundaries; usually only needed on TCP connections.
−p servicename Post the service under srv(3) as /srv/service name.
−u user When connecting to the remote server, log in as user. Since srv Old 9p does no authentication, and since new kernels cannot authenticate to old services, the likeliest value of user is none.
−x command Run command and use its standard input and output as the 9P service connection. If the command string contains blanks, it should be quoted.
−n network−addr Dial network−addr to establish the connection.
−f file Use file (typically an existing srv(3) file) as the connection.
Srv Old 9p is run automatically when a cpu(1) call is received on the service port for the old protocol.

EXAMPLES
To see kremvax’s and deepthought’s files in /n/kremvax and /n/deepthought:
9fs kremvax
9fs hhgttg /n/deepthought
To mount as user none a connection to an older server kgbsun:
srvold9p -u none -m /n/kgbsun -p kgbsun -n il!kgbsun
Other windows may then mount the connection directly:
mount /srv/kgbsun /n/kgbsun
To connect to an instance of the Unix server u9fs(4) started via ssh(1):
srvssh unix

FILES
/srv/* ports to file systems and servers posted by srv and 9fs

SOURCE
/sys/src/cmd/srv.c
/rc/bin/9fs
/rc/bin/srvssh
/sys/src/cmd/srvold9p

SEE ALSO
bind(1), auth(2), dial(2), srv(3), exportfs(4), import(4), ftpfs(4), u9fs(4)

BUGS
Srv does not explicitly report failures of auth_proxy (see auth(2)); mount (see bind(1)) does.
NAME
32vfs, cpiofs, tapfs, tarfs, tpfs, v6fs, v10fs, zipfs – mount archival file systems

SYNOPSIS
fs/32vfs [−b blocksize] [−m mountpoint] [−p passwd] [−g group] file
fs/cpiofs
fs/tapfs
fs/tarfs
fs/tpfs
fs/v6fs
fs/v10fs
fs/zipfs

DESCRIPTION
These commands interpret data from traditional tape or file system formats stored in file, and
mount their contents (read-only) into a Plan 9 file system. The optional −p and −g flags specify
Unix-format password (respectively group) files that give the mapping between the numeric user-
and group-ID numbers on the media and the strings reported by Plan 9 status inquiries. The −m flag introduces the name at which the new file system should be attached; the default is
/n/tapefs.
32vfs interprets raw disk images of 32V systems, which are ca. 1978 research Unix systems for the
VAX (512 byte block size, the default), and also pre-FFS Berkeley VAX systems (1KB block size).
Cpiofs interprets cpio tape images (constructed with cpio’s c flag).
Tarfs interprets tar tape images.
Tpfs interprets tp tapes from the Fifth through Seventh Edition research Unix systems.
Tapfs interprets tap tapes from the pre-Fifth Edition era.
V6fs interprets disk images from the Fifth and Sixth edition research Unix systems (512B block
size).
V10fs interprets disk images from the Tenth Edition research Unix systems (4KB block size).
Zipfs interprets zip archives (see gzip(1)).

SOURCE
These commands are constructed in a highly stereotyped way using the files fs.c and util.c in
/sys/src/cmd/tapefs, which in turn derive substantially from ramfs(4).

SEE ALSO
intro(5), ramfs(4).
NAME
telco, faxreceive, faxsend, fax, telcofax, telcodata - telephone dialer network

SYNOPSIS
telco [-p][-i source-id][-v] dialer−devs
aux/faxsend address page1 ...
aux/faxreceive [-s spool−dir][-v]
fax [-v] telno recipient [ files]
service/telcofax

DESCRIPTION
Telco is a file server that provides a network interface to Hayes telephone dialers. The interface is the same as that provided by ip(3) and can be used by any program that makes network connections using dial(2). The network addresses used by telco are telephone numbers.

The options are
−p   use pulse dialing
−v   verbose: write to the log file all communications with the dialer.
−i   specify a source−id to be used during FAX transfers

Some control of outgoing calls can be encoded in the address. Normally, addresses are of the form telco! number, where number is a decimal telephone number. However, commas in the telephone number can be used to insert pauses in the dialing process. Dialing options can be added to the end of the address, separated by ’!’. The dialing options are
compress  turn on compression (default off)
baudrate   a decimal number representing the highest baud rate with which to make the call
fax        to make a Class 2 facsimile call (used by programs such as faxsend)

Telco also answers incoming calls. Upon receiving a facsimile call, telco starts the script /rc/bin/service/telcofax. For data calls it starts /rc/bin/service/telcodata.
Each is started with the network connection as both standard input and standard output and with two arguments, the file name of the network connection, e.g., /net/telco/0/data, and the type of modem. Currently, the only modem types supported are:

MT1432      Multitech’s 14400 baud modem
MT2834      Multitech’s 28800 baud modem
ATT14400    the 14400 baud modem in Safarils
VOCAL       the 14400 baud Vocal modem

All other modems are assumed to be compatible with the standard Hayes command subset.

Faxreceive is normally started by /rc/bin/service/telcofax. It reads and spools a CCITT Group 3 (G3) encoded FAX, and then starts the script /sys/lib/fax/receiverc, passing it four arguments: the spool file name, Y (for success) or N, the number of pages, and the id string passed by the caller. This script sends by mail(1) notification to a list of recipients kept in the file /mail/faxqueue/faxrecipients; the script and the list should be edited to match local needs. Faxreceive’s options are:
−s   specify a different spool directory; the default is /mail/faxqueue.
−v   verbose: write to the log file all communications with the modem.

Faxsend transmits a FAX to address. Page1 and all arguments that follow are names of files containing G3 encoded FAX images, one per page.

Fax is a shell script that converts to G3 format PostScript, G3, text, or other files acceptable to Ip(1) and queues the result to be transmitted to a FAX machine. A standard cover sheet, derived from /sys/lib/fax/h.ps, is sent before the message. Telno is the destination telephone number. Recipient is the name of the recipient to be placed on the cover sheet. If no files are specified, standard input is converted and sent. The −v option invokes page(1) on the generated G3 files instead of transmitting them via FAX machine.
EXAMPLE
Start the dialer on a PC, then use con to phone out.

```
telco /dev/eia1
con -l telco!18005551212
```

The connection will be made at the highest negotiable baud rate. To use the best negotiable compression scheme as well:

```
con -l telco!18005551212!compress
```

FILES
- `/mail/faxqueue/*`
- `/rc/bin/service/telcodata`
- `/rc/bin/service/telcofax`
- `/sys/log/telco`
- `/sys/lib/fax/receiverc`
- `/mail/faxqueue/faxrecipients`
- `/sys/lib/fax/h.ps`
- `/sys/log/fax`

SOURCE
- `/sys/src/cmd/telco/*`
- `/sys/src/cmd/fax/*`

SEE ALSO
`con(1), ip(3)`

BUGS
These programs require the Class 2 facsimile interface. This means that `faxsend` and `faxreceive` will not work on most portable computers since they have Class 1 interfaces.

The modem specific information is currently built into the source. This should be in a user modifiable file.
NAME
u9fs – serve 9P from Unix

SYNOPSIS
u9fs [ −Dn] [ −a authtype ] [ −A autharg ] [ −l logfile ] [ −m msize ] [ −u onlyuser ] fsroot

DESCRIPTION
U9fs is not a Plan 9 program. Instead it is a program that serves Unix files to Plan 9 machines using the 9P protocol (see intro(5)). It is typically invoked on a Unix machine by inetd with its standard input and output connected to a network connection, typically TCP on an Ethernet. It typically runs as user root and multiplexes access to multiple Plan 9 clients over the single wire. It assumes Plan 9 uids match Unix login names, and changes to the corresponding Unix effective uid when processing requests. Characters in file and directory names unacceptable to Plan 9 are translated into a three-character sequence: \ followed by two hexadecimal digits. U9fs serves both 9P1 (the 9P protocol as used by the second and third editions of Plan 9) and 9P2000.

The options are:
−D Write very chatty debugging output to the log file (see −l option below).
−n Signals that u9fs is not being invoked with a network connection on standard input and output, and thus should not try to determine the remote address of the connection. This is useful when u9fs is not invoked from inetd (see examples below).
−z Truncate the log file on startup. This is useful mainly when debugging with −D.
−a authtype Sets the authentication method to be used. Authtype should be rhosts, none, or p9any. The default is rhosts, which uses the ruserok library call to authenticate users by entries in /etc/hosts.equiv or $HOME/.rhosts. This default is discouraged for all but the most controlled networks. Specifying none turns off authentication altogether. This is useful when u9fs is not invoked from inetd (see examples below, or srsvsh in srv(4)). Specifying p9any uses the fourth edition Plan 9 authentication mechanisms. The file /etc/u9fs.key, or autharg if specified (see the −A option), is consulted for the authentication data and should be suitably protected. This file must contain exactly three lines: secret (plaintext password), u9fs−user (user id), and plan9−auth.dom (authentication domain).

Finally, factotum must be taught a key of the form:
key proto=p9sk1 dom=plan9−auth.dom user=u9fs−user !password=secret

−A autharg Used to specify an argument to the authentication method. See the authentication descriptions above.
−l logfile Specifies the file which should contain debugging output and other messages. The out−of−the−box compile−time default is /tmp/u9fs.log.
−m msize Set msize for 9P2000 (see open(5)).
−u user Treat all attaches as coming from user. This is useful in some cases when running without inetd; see the examples.

If fsroot is specified, u9fs will serve only that tree; othewise, it will serve the entire Unix file system.

EXAMPLES
Plan 9 calls 9P file service 9fs with TCP port number 564. Set up this way on a machine called, say, kremvax, u9fs may be connected to the name space of a Plan 9 process by
9fs kremvax

For more information on this procedure, see srv(4) and bind(1).

By default, u9fs serves the entire file system of the Unix machine. It forbids access to devices because the program is single−threaded and may block unpredictably. Using the attach specifier device connects to a file system identical to the usual system except it only permits device access (and may block unpredictably):
srv tcp!kremvax!9fs
mount −c /srv/tcp!kremvax!9fs /n/kremvax device
(The 9fs command does not accept an attach specifier.) Even so, device access may produce unpredictable results if the block size of the device is greater than 8192, the maximum data size of a 9P message.

The source to u9fs is in the Plan 9 directory /sys/src/cmd/unix/u9fs. To install u9fs on a Unix system with an ANSI C compiler, copy the source to a directory on that system and run make. Then install the binary in /usr/etc/u9fs. Add this line to inetd.conf:

```
  9fs  stream  tcp  nowait  root /usr/etc/u9fs u9fs
```

and this to services:

```
  9fs  564/tcp  9fs  # Plan 9 fs
```

Due to a bug in their IP software, some systems will not accept the service name 9fs, thinking it a service number because of the initial digit. If so, run the service as u9fs or 564.

On systems where listeners cannot be started, execnet(4) is useful for running u9fs via other network mechanisms; the script srvssh in srv(4) provides this for the ssh protocol.

**SOURCE**

/sys/src/cmd/unix/u9fs

**DIAGNOSTICS**

Problems are reported to the log file specified with the \(-l\) option (default /tmp/u9fs.log). The \(-D\) flag enables chatty debugging.

**SEE ALSO**

bind(1), execnet(4), srv(4), ip(3), nfsserver(8)

**BUGS**

The implementation of devices is unsatisfactory.

Semantics like remove-on-close or the atomicity of wstat are hard to provide exactly.
NAME
upasfs, startupasfs - mail file server

SYNOPSIS
upas/fs [-f mailbox] [-bnps] [-m mntpoint]
startupasfs

DESCRIPTION
Fs is a user level file system that reads mailboxes and presents them as a file system. A user normally starts fs in his/her profile after starting plumber(4) and before starting a window system, such as ria(1) or acme(1). The file system is used by nedmail and acme(1)'s mail reader to parse messages. Fs also generates plumbing messages used by biff and faces(1) to provide mail announcements.

Startupasfs is a shell script suitable for use in one's profile. It runs fs -s for the invoking user if none is already running, and always mounts the user's posted fs on /mail/fs.

The mailbox itself becomes a directory under /mail/fs. Each message in the mailbox becomes a numbered directory in the mailbox directory, and each attachment becomes a numbered directory in the message directory. Since an attachment may itself be a mail message, this structure can recurse ad nauseam.

Each message and attachment directory contains the files:

body the message minus the RFC822 style headers
cc the address(es) from the CC: header
date the date in the message, or if none, the time of delivery
digest an SHA1 digest of the message contents
disposition inline or file
filename a name to use to file an attachment
from the from address in the From: header, or if none, the address on the envelope.
header the RFC822 headers
info described below, essentially a summary of the header info
inreplyto contents of the in-reply-to: header
mimeheader the mime headers
raw the undecoded MIME message
rawbody the undecoded message body
rawheader the undecoded message header
replyto the address to send any replies to.
sender address
recipient addresses SHA1 digest
cc addresses bcc addresses
envelope date RFC822 date
envelope date RFC822 date
subject the contents of the subject line
to the address(es) from the To: line.
type the MIME content type
unixheader the envelope header from the mailbox

The info file contains the following information, one item per line. Lists of addresses are single-space separated.

Deleting message directories causes the message to be removed from the mailbox.

The mailbox is reread and the structure updated whenever the mailbox changes. Message directories are not renumbered.

The file /mail/fs/ctl is used to direct fs to open/close new mailboxes or to delete groups of messages atomically. The messages that can be written to this file are:
open path mboxname opens a new mailbox. path is the file to open, and mboxname is the name that appears under /mail/fs.
close mboxname close mboxname. The close takes affect only after all files open under /mail/fs/mboxname have been closed.
delete mboxname number ... Delete the messages with the given numbers from mboxname.
The options are:
  -f file use file as the mailbox instead of the default, /mail/box/username/mbox.
  -b stands for biffing. Each time new mail is received, a message is printed to standard output containing the sender address, subject, and number of bytes. It is intended for people telnetting in who want mail announcements.
  -n Don't open a mailbox initially. Overridden by -f.
  -p turn off plumbing. Unless this is specified, fs sends a message to the plumb port, seeming, from source mailfs's for each message received or deleted. The message contains the attributes sender=<contents of from file>, mailtype=deleted or new, and length=<message length in bytes>. The contents of the message is the full path name of the directory representing the message.
  -s causes fs to post itself in /srv with a name of the form /srv/upasfs. user.
  -m specifies a mount point other than /mail/fs.
Fs will exit once all references to its directory have disappeared.

Fs interprets mailbox file names of the form /proto/host/user to mean access an account on host using the given protocol. Authentication is delegated to factotum(4). The final /user may be omitted, in which case the user name is gleaned from the key held by factotum. The following protocols are supported:

pop cleartext POP with password authentication
apop cleartext POP with challenge-response (APOP) authentication
pops
poptls TLS-encrypted POP with password authentication
apopls
apopls
imap cleartext IMAP
imaps TLS-encrypted IMAP

The two IMAP protocols allow an optional fourth field specifying a mailbox name, for example /imap/server/user/stored.

Poptls and apopls connect to port 110 in plaintext and start TLS using the POP STLS command. Pops and apops connect to port 995 and start TLS before initiating the POP conversation. Imaps connects to port 993 and starts TLS before initiating the IMAP conversation. There should probably be an imapTLS protocol as well. (ImapTLS would connect to port 143 in plaintext and start TLS using the IMAP STARTTLS command. (That's the nice thing about standards—there's so many to choose from.))

FILES
/mail/box/* mail directories
/mail/box/*/mbox mailbox files
/mail/box/*/L.reading mutual exclusion lock for multiple mbox readers
/mail/box/*/L.mbox mutual exclusion lock for altering mbox

SOURCE
/sys/src/cmd/upas/fs
/rc/bin/startupasfs

SEE ALSO
aliasmail(8), faces(1), filter(1), mail(1), marshal(1), mlimgr(1), nedmail(1), qer(8), rewrite(6), send(8), upasfs(4)
NAME
audio, ccid, disk, ether, kb, print, probe, serial, useject, usbfat: - Universal Serial Bus device drivers

SYNOPSIS
usb/kb [-dkm] [-a accel] [dev ...]
usb/disk [-Dd] [-m mnt] [-s srv] [dev ...]
usbfat: [disk ...]
usbeject [disk ...]
usb/audio [-dpV] [-m mnt] [-s srv] [-v vol] [dev]
usb/ether [-Dd] [-m mnt] [-s srv] [dev ...]
usb/serial [-Dd] [-m mnt] [-s srv] [dev ...]
usb/print [-d] [dev ...]
usb/ccid [-d] usb/probe

DESCRIPTION
These programs drive USB devices of specific classes via usb(3). Usually they are started by usbd(4) upon attachment of the device to the bus. Less often, users start them manually, depending on usb(4)'s configuration. Usually, kb and disk are started by usbd and other programs are started by hand.

Without arguments, the drivers handle all the devices (of the appropriate USB class) found on the bus. To make a driver handle only certain devices, supply as arguments the paths for the directories of the devices (actually of their zero endpoints).

Drivers that provide file systems accept options -s and -m to instruct them to post a 9P connection at srv(3) with the given name and/or to mount themselves at mnt. When embedded into usbd these options may not be used. In this case, the file tree supplied by the device driver is available through the file system provided by usbd, usually mounted at /dev and reachable through the 9P connection posted at /srv/usb.

Options -d and -D present on most drivers trigger debug diagnostics and file system debugging diagnostics. Repeating any one of these may increase verbosity.

To help locate devices of interest, probe lists all the USB devices available, including those with no driver started.

Keyboards and mice
kb supports USB keyboards and mice either as separate USB devices or as a single combined USB device. Scan codes from the keyboard are sent to /dev/kbin to let the kernel process them. Mouse events are sent to /dev/mousein in the same way.

The following options are understood:
-a Accelerate the mouse to level n (similar to the kernel mouse driver acceleration).
-k Serve just the keyboard (and not the mouse).
-m Serve just the mouse (and not the keyboard).

Disks
disk configures and manages USB mass storage devices. It provides a file system (usually seen at /dev) that includes one directory per storage device, named sduN. M in correspondence with the usb device number and the storage unit number (or LUN). For example, LUN number 2 on /dev/usb/ep3.0 can be accessed through /dev/sdU3.2.

The storage device directory contains the usual files served by sd(3): data, raw, and ctl.

The ctl file supplies the device geometry when read.

The script usbfat: mounts the FAT file systems in the DOS partitions of the named disks; if none, it mounts those file systems found at /dev/sdU*.*/data. When more than one partition is found, a suffix is appended to the disk name to identify the partition number. The script usbeject undoes the effect. If no argument is given, it unmounts all USB disks. An argument sduN unmounts all partitions from disk with USB target N.
Printers
Print provides a single file can be written to print on a USB printer. Options are similar to those of
disk. The file is also bound at /dev/lp as is customary.

Ethernet adapters
Ether provides a file interface similar to that of ether(3) for each USB Ethernet adapter found. The
name of an Ethernet device is etherUN where N is the device name. When started manually, the
file interface is mounted at /net as is customary.

Serial and JTAG ports
Serial provides a file system (usually mounted at /dev) that includes one directory per USB serial
port, named eiaUN or eiaUN. M. In this directory there are two files, eiaU, similar to eiaN in
uart(3), and eiaUctl, which admits writes in the same format as eiaNctl in uart(3). Reading
from eiaUctl gives the serial port’s settings in the same format as eiaNstatus in uart(3). Options are similar to those of disk.

JTAG ports are similar but the files are named jtag and jtagctl.

Audio devices
Usbaudio configures and manages a USB audio device. It implements a file system, normally
mounted on /dev, but this can be changed with -m, containing files volume, audioctl, audioin. The names volume and audio maintain backward compatibility with
the Soundblaster driver.

The -V option (verbose) causes audio to print information about the device on startup. The -s
option specifies a name for a file descriptor to be posted in /srv. The -v options sets initial
volume.

Reading volume or audioctl yields the device’s settings. The data format of volume is com-
patible with the Soundblaster and produces output in this format:

```
    audio out 65
    treb out 0
    bass out 0
    speed out 44100
```

This file can be written using the same syntax. The keyword out may be omitted. Settings are
given as percentages of the range, except for speed which is in Hz.

The file audioctl provides more information, using up to 6 columns of 12 characters each.
From left to right, the fields are: control name, in or out, current value, minimum value,
maximum, and resolution. There are 3, 5, or 6 columns present. Maxima and resolution are omit-
ted when they are not available or not applicable. The resolution for speed is reported as 1 (one) if
the sampling frequency is continuously variable. It is absent if it is settable at a fixed number of
discrete values only.

When all values from audioctl have been read, a zero-length buffer is returned (the usual end-
of-file indication). A new read will then block until one of the settings changes, then report its
new value.

The file audioctl can be written like volume.

Audio data is written to audio and read from audioin. The data format is little-endian, sam-
ple ordered primarily by time and secondarily by channel. Samples occupy the minimum integral
number of bytes. Read and write operations of arbitrary size are allowed.

Ccids
Ccids discovers and configures SIM or SAM cards using the CCID standard. It provides a file system
(usually mounted at /dev) that includes three files, ctl, raw and rpc. Reading from ctl a
description of the smartcard reader capabilities is printed. raw is just intended for debugging.
Reads and writes to the raw file send and receive raw CCID packets. Smart cards identify them-
sews by giving out an ATR, an array of characters describing the card uniquely. Users of the
driver write the ATR to the rpc file and are blocked until a card with that ATR is seen. From then
on they can do ICC RPCs using whatever language the smart card speaks. A small write cancels an
outstanding RPC.

The driver takes care of powering the card adequately, based on its ATR, and tunnelling the RPCs
through the USB device. Only slot 0 is supported.
When the smartcard disappears, all reads and write fail until the file is reopened and a new ATR is written to it.

SOURCE
/sys/src/cmd/usb

SEE ALSO
kbin(3), mouse(3), sd(3), uart(3), usb(3), usbd(4), partfs(8)

BUGS
The various device drivers are generic USB drivers and may work only for certain devices on each class.
ATA storage devices are not supported.
The Ethernet device works only for certain ASIX-based cards and for CDC devices. Both the Ethernet and printer drivers have not been tested and it is likely they will fail.
The serial driver works only for the Prolific chip and Ftdi, and control of the dcd and dsr signals and some of the extra features are unimplemented. For Ftdi, only the Sheevaplug and Guruplug have been tried. There is support for the EHCI debug port, but it loses bytes.
The entire set of drivers is new and therefore potentially unreliable. A list of working devices must be compiled.
NAME
usbd - Universal Serial Bus daemon

SYNOPSIS
usbd [-Dd] [-s srv] [-m mnt] [ hub... ]

DESCRIPTION
Usbd complements usb(3) to provide USB I/O for device drivers. It enumerates the bus, polling hub ports to detect device attachments and detachments, performs initial configuration of setup endpoints, and writes extra information into usb(3) endpoint control files, to ease device location.

By default, usbd opens all setup endpoints found at /sys/usb (which correspond to built-in hubs initialized by the kernel during boot). Paths to directories representing setup endpoints for hubs can be given as arguments to restrict usbd operation to such hubs.

When a device is attached, depending upon a configuration file compiled into usbd, the appropriate device driver may be started without user intervention. This mechanism can be used to statically link some USB device drivers into usbd itself. Initial configuration for setup endpoints is performed independently of this configuration.

Usbd provides a file interface used to change debugging flags, and also used by USB device drivers statically linked into usbd. By default, the file system is mounted (after) at /dev and a 9P connection is posted at /srv/usb.

Besides files provided by device drivers, the file usbdctl is always present in the file interface. It accepts these control requests:

disable n  Sets the debugging level to n.
disable n  Sets the file system debugging level to n.
dump      Prints the list of devices and file systems known by usbd.

Usbd recognizes the following options:

-d         Print debugging diagnostics. Repeating the option increases verbosity.
-D         Print debugging diagnostics for the file system interface.
-m mnt     Mount the served file system at mnt.
-s srv     Post a 9P connection at #s/srv.

Configuration
Usbd can be configured to start drivers for devices matching one or more CSPs (hex representation of USB class, subclass and protocol, class, subclass, protocol, vendor id, or device id. When a new device is attached, usbd scans the configuration and, if an entry matches the device descriptor, starts the driver. If no driver is configured, the setup endpoint for the device is left configured to let the user start the driver by hand.

Configuration is via compilation because one of the options is to embed (link) the driver into the usbd binary. If the driver is embedded, usbd creates a process for it and calls its main entry point. Otherwise, usbd tries to locate the driver binary in /bin/usb and creates a process to execute it.

The configuration file, usbd, has two sections: embed and auto. Each section includes lines to configure particular drivers. A driver may have more than one line if necessary. Each line includes the name of the driver (the base name of the binary) and one or more attributes of the form

name=value

The following attributes exist:

class Value may be the name of the class or a number identifying the device class (using C syntax). The following class names are known: audio, comms, hid, printer, storage, hub, and data.

subclass Value is the number of the device subclass.

proto Value is the number of the device protocol.

csp Value is the hexadecimal number describing the CSP for the device.

vid Value is the vendor id.

vid Value is the device id.

args This must be the last field. The value is the rest of the line, and is supplied as arguments to the driver process.
Several environment variables can be used to alter the behaviour of usbd, for example, for use in plan9.ini(8). usbddebug sets a debug level (zero for no diagnostics and positive values for increasing verbosity). kbargs overrides the keyboard arguments as specified by the configuration file. diskargs overrides the disk arguments in the same way.

EXAMPLE
This configuration file links usb/kb into usbd when it is compiled. It arranges for the driver’s entry point, kbmain in this case, to be called for any device with CSPs matching either 0x010103 or 0x020103. Option –d will be supplied as command line arguments for kbmain. This configuration also arranges for /bin/usb/disk to start (with no arguments) whenever a device of class storage is attached.

    embed
    kb    csp=0x010103 csp=0x020103    args=−d
    auto
        disk class=storage    args=

FILES
/srv/usb  9P connection to the driver file system.
/dev     mount point for the driver file system.
/sys/src/cmd/usb/usbd/usbd
disk Configuration file deciding which devices are included into usbd and which ones are started automatically.

SOURCE
/sys/src/cmd/usb/usbd

SEE ALSO
usb(2), usb(3), usb(4)

BUGS
Usbd is not supposed to be restarted. This is arguable.
Not heavily exercised yet.
NAME
vacfs - a Venti-based file system

SYNOPSIS
vacfs [-dips] [-c cachessize] [-h host] [-m mtpt] [-S srvname] vacfile

DESCRIPTION
Vacfs interprets the file system created by vac(1) so that it can be mounted into a Plan 9 file hierarchy. The data for the file system is stored on venti(8) with a root fingerprint specified in vacfile. Vacfs is currently rather limited: access is read-only, clients are not authenticated, and groups are assumed to contain a single member with the same name. These restrictions should eventually be removed.

Options to vacfs are:

- c cachessize The number of file system blocks to cache in memory. The default is 1000 blocks.
- d Print debugging information to standard error.
- h host The network address of the Venti server. The default is taken from the environment variable venti. If this variable does not exist, then the default is the metaname $venti, which can be configured via ndb(6).
- i Use file descriptors 0 and 1 as the 9P communication channel rather than create a pipe.
- m mtpt The location to mount the file system. The default is /n/vac.
- p Disables permission checking.
- s Post the 9P channel in /srv/vacfs rather than mounting it on mtpt.
- S srvname Post the 9P channel in /srv/srvname rather than mounting it on mtpt.

SOURCE
/sys/src/cmd/vac

SEE ALSO
vac(1), venti(8)
NAME
webcookies - HTTP cookie manager

SYNOPSIS
webcookies [-f cookiefile][-m mtpt][-s service]

DESCRIPTION
Webcookies manages a set of HTTP cookies, which are used to associate HTTP requests with persistent state (such as user profiles) on many web servers. Webcookies reads cookiefile (default $home/lib/webcookies) and mounts itself at mtpt (default /mnt/webcookies). If service is specified, cookies will post a service file descriptor in /srv/service.

The cookie file contains one cookie per line; each cookie comprises some number of attr=value pairs. Cookie attributes are:

name=value The name of the cookie on the remote server.
value=value The value associated with that name on the remote server. The actual data included when a cookie is sent back to the server is "name=value" (where, confusingly, name and value are the values associated with the name and value attributes.

domain=domain The domain within which the cookie can be used. If domain is an IP address, the cookie can only be used when connecting to a web server at that IP address. If domain is a pattern beginning with a dot, the cookie can only be used for servers whose name has domain as a suffix. For example, a cookie with domain=.bell-labs.com may be used on the web sites www.bell-labs.com and www.research.bell-labs.com.

path=path The cookie can only be used for URLs with a path (the part after http://hostname) beginning with path.

version=version The version of the HTTP cookie specification, specified by the server.
comment=comment A comment, specified by the server.
expire=expire The cookie expires at time expire, which is a decimal number of seconds since the epoch.
secure=1 The cookie may only be used over secure (https) connections.
explicitdomain=1 The domain associated with this cookie was set by the server (rather than inferred from a URL).
explicitpath=1 The path associated with this cookie was set by the server (rather than inferred from a URL).

netscapestyle=1 The server presented the cookie in "Netscape style," which does not conform to the cookie standard, RFC2109. It is assumed that when presenting the cookie to the server, it must be sent back in Netscape style as well.

Webcookies serves a directory containing two files. The first, cookies, is a textual representation of the cookie file, which can be edited to change the set of cookies currently held. The second, http, is intended to be used by HTTP clients to access cookies. Upon opening http, the client must write a full URL to it. After writing the URL, reading from the file will yield any HTTP Cookie: headers that should be included in the request for this particular URL. Once the request has been made, any Set-Cookie: lines in the HTTP response header should be written to the file to save them for next time. If cookies decides not to accept the cookie (as outlined in RFC2109, section 4.3.4), no indication is given.
Hget() uses /mnt/webcookies/http, when it exists, to manage cookie state. Webfs does not (yet).

SOURCE
/sys/src/cmd/webcookies.c
SEE ALSO

*hget*(1)

BUGS

It’s not clear what the relationship between *cookiefs* and something like *webfs* should be.
NAME
  webfs - world wide web file system

SYNOPSIS
  webfs [-c cookiefile] [-m mtpt] [-s service] ...

DESCRIPTION
  Webfs presents a file system interface to the parsing and retrieving of URLs. Webfs mounts itself at
  mtpt (default /mnt/web), and, if service is specified, will post a service file descriptor in
  /srv/service.

  Webfs presents a three-level file system suggestive of the network protocol hierarchies ip(3) and
  ether(3).

  The top level contains three files: ctl, cookies, and clone.

  The ctl file is used to maintain parameters global to the instance of webfs. Reading the ctl file
  yields the current values of the parameters. Writing strings of the form "attr value" sets a
  particular attribute. Attributes are:

    chatty9p
    The chatty9p flag used by the 9P library, discussed in 9p(2). 0 is no debugging, 1 prints
    9P message traces on standard error, and values above 1 present more debugging, at the
    whim of the library. The default for this and the following debug flags is 0.

    fsdebug
    This variable is the level of debugging output about the file system module.

    cookiedebug
    This variable is the level of debugging output about the cookie module.

    urldebug
    This variable is the level of debugging output about URL parsing.

    acceptcookies
    This flag controls whether to accept cookies presented by remote web servers. (Cookies
    are described below, in the discussion of the cookies file.) The values on and off are
    synonymous with 1 and 0. The default is on.

    sendcookies
    This flag controls whether to present stored cookies to remote web servers. The default is
    on.

    redirectlimit
    Web servers can respond to a request with a message redirecting to another page. Webfs
    makes no effort to determine whether it is in an infinite redirect loop. Instead, it gives up
    after this many redirects. The default is 10.

    useragent
    Webfs sends the value of this attribute in its User-Agent: header in its HTTP requests.
    The default is "webfs/2.0 (plan 9)."

  The top-level directory also contains numbered directories corresponding to connections, which
  may be used to fetch a single URL. To allocate a connection, open the clone file and read a num
  ber n from it. After opening, the clone file is equivalent to the file n/ctl. A connection is
  assumed closed once all files in its directory have been closed, and is then will be reallocated.

  Each connection has its own private set of acceptcookies, sendcookies, redirectlimit, and useragent variables, initialized to the defaults set in the root's ctl
  file. The per-connection ctl file allows editing the variables for this particular connection.

  Each connection also has a URL string variable url associated with it. This URL may be an abso
  lute URL such as http://www.lucent.com/index.html or a relative URL such as ../index.html. The
  baseurl string variable sets the URL against which relative URLs are interpreted. Once the URL
  has been set, its pieces can be retrieved via individual files in the parsed directory. Webfs parses
  the following URL syntaxes; names in italics are the names of files in the parsed directory.
scheme: schemedata
http://host/path[?query][#fragment]
ftp://[user[:password]@]host/path[:type=ftptype]
file:path

If there is associated data to be posted with the request, it can be written to postbody. Finally, opening body initiates the request. The resulting data may be read from body as it arrives. After the request has been executed, the MIME content type may be read from the contenttype file.

The top-level cookies file contains the internal set of HTTP cookies, which are used by HTTP servers to associate requests with persistent state such as user profiles. It may be edited as an ordinary text file. Multiple instances of webfs and webcookies(4) share cookies by keeping their internal set consistent with the cookiefile (default $home/lib/webcookies), which has the same format.

These files contain one line per cookie; each cookie comprises some number of attr=value pairs. Cookie attributes are:

name=name
The name of the cookie on the remote server.

value=value
The value associated with that name on the remote server. The actual data included when a cookie is sent back to the server is "name=value" (where, confusingly, name and value are the values associated with the name and value attributes.

domain=domain
If domain is an IP address, the cookie can only be used for URLs with host equal to that IP address. Otherwise, domain must be a pattern beginning with a dot, and the cookie can only be used for URLs with a host having domain as a suffix. For example, a cookie with domain=.bell-labs.com may be used on hosts www.bell-labs.com and www.research.bell-labs.com (but not www.not-bell-labs.com).

path=path
The cookie can only be used for URLs with a path beginning with path.

version=version
The version of the HTTP cookie specification, specified by the server.

comment=comment
A comment, specified by the server.

expire=expire
The cookie expires at time expire, which is a decimal number of seconds since the epoch.

secure=1
The cookie may only be used over secure (https) connections. Secure connections are currently unimplemented.

explicitdomain=1
The domain associated with this cookie was set by the server (rather than inferred from a URL).

explicitpath=1
The path associated with this cookie was set by the server (rather than inferred from a URL).

netscapestyle=1
The server presented the cookie in "Netscape style," which does not conform to the cookie standard, RFC2109. It is assumed that when presenting the cookie to the server, it must be sent back in Netscape style as well.

EXAMPLE
/sys/src/cmd/webfs/webget.c is a simple client.

SOURCE
/sys/src/cmd/webfs
SEE ALSO
hget(1), webcookies(4)

BUGS
It's not clear what the relationship between hget, webcookies and webfs should be.
NAME
wikifs, wikipost − wiki file system

SYNOPSIS
wikifs [-DM] [-a announce]... [-m mtpt] [-p perm] [-s service] dir
version uri [search]

DESCRIPTION
A wiki is a web server that facilitates easy editing of the pages it contains. Wikifs presents a wiki in
two forms: as web pages to be served via httpd(8) and as text files to be viewed via the acme(1)
wiki client (see /mnt/wiki/guide).

Wikifs presents a file system interface to the wiki data stored in dir. By default, wikifs mounts itself
at /mnt/wiki: the −m flag specifies a different mount point, and the −M flag causes wikifs not to
mount at all. Wikifs also announces 9P network services on the addresses given as arguments to
−a options. If the −s option is given, wikifs will post a service file descriptor in /srv/service
with permission perm (default 600). The −D flag causes a transcript of the 9P conversation to be
written to standard error.

The wiki holds both the current pages and also all versions of all pages that have ever existed. All
pages have time stamps associated with them. When a user wants to edit a page, he reads the cur-
rent page from the wiki, noting the time stamp on the page. When a user writes changes to a
page, he includes the time stamp of the page he started with. If the page has been updated by
someone else while he was editing, the write will fail. This is called a “conflicting write.” The sub-
mission is still saved in the history, so that the user can compare the page he submitted with the
changes that were made while he was editing.

Each version of each page is described by a text file containing one or more metadata lines fol-
lowed by the page contents. The metadata lines begin with a capital letter specifying the type of
data. Currently the metadata types are:

D The date this page was written, in decimal seconds since the epoch.
A The author of this version of the page. Typically the rest of the line takes the form name
ip−address.
X This page’s contents were submitted but rejected due to a conflicting write.

After the metadata comes the actual page contents; each line of page contents is prefixed with a #
character.

The directory dir/d contains all the wiki data. Typically it is world−writable so that wikifs can run
as none. Each page on the wiki has a unique sequence number n; for each page, the d directory
contains three files n, n.hist, and L.n. The file n holds the current version of the page: the first
line of n is the page title, followed by page metadata and contents as described above. The
append-only file n.hist holds the history of the page. The first line of n.hist is the title of
the page. The rest of the file is the metadata and contents of every version of the page that has
been submitted to the wiki. L.n is a lock file for the page: it must be held while reading or writing
n and n.hist. The lock files allow multiple instances of wikifs to coexist peacefully. Finally, the
map file (with associated lock L.map) provides a mapping from sequence numbers to page
titles. Each map line is a decimal n, a single space, and then the title. Since titles are presented as
names by wikifs, they cannot contain slashes.

Wikifs presents a three−level file system. The top level contains per−page directories named by the
page titles with spaces turned into underscores. Each page also has a number associated with it
(see the discussion of the wiki data files below). The number corresponding to a page may also be
used to access it, although directory listings will always present the title. The new file is used to
add new or revised pages to the wiki: writes to the file should be in the usual textual format: a title
line, metadata lines, and page contents. Once all the contents have been written, a final zero−
length message should be written to mark the end of the page. This last write will return an error
if a conflicting write has occurred. After writing the file, the client may read from new to obtain
the canonical title for the page, as presented by the file system.
The page directories contain subdirectories representing the history of the page, named by the
decimal time stamp corresponding to each version. In addition to these history directories, the
page directories contain the following files:

current
   The current raw data file for the page.
diff.html
   A web page listing the contents of every version of the page that has ever appeared on the
   wiki. The text is grey by default; differences between versions appear in black.
edit.html
   A web form for editing the current version of the page.
history.html
   A web page listing the time stamps of the historical versions of the page. Each time stamp
   links to a page showing just that version.
history.txt
   A textual formatting of the history. Each time stamp is prefixed with the name of the direc-
tory corresponding to that version.
index.html
   An HTML formatting of the current version of the page.
index.txt
   A textual formatting of the current version of the page.
werror.html
   An HTML error page to be returned by wikipost on conflicting writes.

The HTML files are generated from the templates with the same names in dir, except that
index.html and index.txt are generated from the templates page.html and page.txt.
The history directories are similar to the page directories but only contain current,
index.html, and index.txt. This index.html and index.txt are generated from the
templates oldpage.html and oldpage.txt.
The httpd(8) helper program wikipost is used to process editing requests posted to the web server
by users. It expects the posted form to contain these (usually hidden) fields: TITLE, the title of
the page; VERSION, the time stamp of the page that is being edited; service, the service name
associated with this wiki (wikipost looks for /srv/wiki.service); and base, the base for wiki
URLs in the response.

After mounting the wiki, wikipost writes a page update request to /mnt/wiki/new and then returns
the contents of one HTML file in /mnt/wiki/title. If the write succeeds, wikipost returns
index.html. If the write fails due to a conflicting write, wikipost returns werror.html.

EXAMPLE

The Plan 9 wiki at Bell Labs is started by running:

    wikifs -p 666 -s wiki.plan9 -a tcp!*!wiki /sys/lib/wiki

The wiki is mounted for httpd(8) by an entry in /lib/namespace.httpd:

    # wiki
    mount -b $s/wiki.plan9 /usr/web/wiki/plan9

Notice that the wiki service was explicitly posted with mode 666 so that httpd (running as none)
would be able to mount it.

In the Plan 9 distribution, the directory /sys/lib/wiki contains sample files similar to those
used to start the current Plan 9 wiki.

SOURCE

/sys/src/cmd/wikifs
/sys/src/cmd/ip/httpd/wikipost.c

SEE ALSO

/acme/wiki/guide
NAME
intro - introduction to the Plan 9 File Protocol, 9P

SYNOPSIS
#include <fcall.h>

DESCRIPTION
A Plan 9 server is an agent that provides one or more hierarchical file systems — file trees — that may be accessed by Plan 9 processes. A server responds to requests by clients to navigate the hierarchy, and to create, remove, read, and write files. The prototypical server is a separate machine that stores large numbers of user files on permanent media; such a machine is called, somewhat confusingly, a file server. Another possibility for a server is to synthesize files on demand, perhaps based on information on data structures inside the kernel; the proc(3) kernel device is a part of the Plan 9 kernel that does this. User programs can also act as servers.

A connection to a server is a bidirectional communication path from the client to the server. There may be a single client or multiple clients sharing the same connection. A server’s file tree is attached to a process group’s name space by bind(2) and mount calls; see intro(2). Processes in the group are then clients of the server: system calls operating on files are translated into requests and responses transmitted on the connection to the appropriate service.

The Plan 9 File Protocol, 9P, is used for messages between clients and servers. A client transmits requests (T-messages) to a server, which subsequently returns replies (R-messages) to the client. The combined acts of transmitting (receiving) a request of a particular type, and receiving (transmitting) its reply is called a transaction of that type.

Each message consists of a sequence of bytes. Two-, four-, and eight-byte fields hold unsigned integers represented in little-endian order (least significant byte first). Data items of larger or variable lengths are represented by a two-byte field specifying a count, n, followed by n bytes of data. Text strings are represented this way, with the text itself stored as a UTF-8 encoded sequence of Unicode characters (see utf(6)). Text strings in 9P messages are not NUL-terminated: n counts the bytes of UTF-8 data, which include no final zero byte. The NUL character is illegal in all text strings in 9P, and is therefore excluded from file names, user names, and so on.

Each 9P message begins with a four-byte size field specifying the length in bytes of the complete message including the four bytes of the size field itself. The next byte is the message type, one of the constants in the enumeration in the include file <fcall.h>. The next two bytes are an identifying tag, described below. The remaining bytes are parameters of different sizes. In the message descriptions, the number of bytes in a field is given in brackets after the field name. The notation parameter[n] where n is not a constant represents a variable-length parameter: n[2] followed by n bytes of data forming the parameter. The notation string[s] (using a literal s character) is shorthand for s[2] followed by s bytes of UTF-8 text. (Systems may choose to reduce the set of legal characters to reduce syntactic problems, for example to remove slashes from name components, but the protocol has no such restriction. Plan 9 names may contain any printable character (that is, any character outside hexadecim al 00–1F and 80–9F except slash.) Messages are transported in byte form to allow for machine independence; fcall(2) describes routines that convert to and from this form into a machine-dependent C structure.

MESSAGES

Each T-message has a tag field, chosen and used by the client to identify the message. The reply to the message will have the same tag. Clients must arrange that no two outstanding messages on the same connection have the same tag. An exception is the tag NOTAG, defined as (ushort) -0 in <fcall.h>: the client can use it, when establishing a connection, to override tag matching in version messages.

The type of an R-message will either be one greater than the type of the corresponding T-message or Rerror, indicating that the request failed. In the latter case, the ename field contains a string describing the reason for failure.

The version message identifies the version of the protocol and indicates the maximum message size the system is prepared to handle. It also initializes the connection and aborts all outstanding I/O on the connection. The set of messages between version requests is called a session.

Most T-messages contain a fid, a 32-bit unsigned integer that the client uses to identify a “current file” on the server. Fids are somewhat like file descriptors in a user process, but they are not restricted to files open for I/O: directories being examined, files being accessed by stat(2) calls, and so on — all files being manipulated by the operating system — are identified by fids. Fids are chosen by the client. All requests on a connection share the same fid space; when several clients share a connection, the agent managing the sharing must arrange that no two clients choose the same fid.

The fid supplied in an attach message will be taken by the server to refer to the root of the served file tree. The attach identifies the user to the server and may specify a particular file tree served by the server (for those that supply more than one).

Permission to attach to the service is proven by providing a special fid, called afid, in the attach message. This afid is established by exchanging auth messages and subsequently manipulated using read and write messages to exchange authentication information not defined explicitly by 9P. Once the authentication protocol is complete, the afid is presented in the attach to permit the user to access the service.

A walk message causes the server to change the current file associated with a fid to be a file in the directory that is the old current file, or one of its subdirectories. Walk returns a new fid that refers to the resulting file. Usually, a client maintains a fid for the root, and navigates by walks from the root fid.

A client can send multiple T-messages without waiting for the corresponding R-messages, but all outstanding T-messages must specify different tags. The server may delay the response to a request and respond to later ones; this is sometimes necessary, for example when the client reads from a file that the server synthesizes from external events such as keyboard characters.
Replies (R-messages) to auth, attach, walk, open, and create requests convey a qid field back to the client. The qid represents the server’s unique identification for the file being accessed: two files on the same server hierarchy are the same if and only if their qids are the same. (The client may have multiple fids pointing to a single file on a server and hence having a single qid.) The thirteen-byte qid fields hold a one-byte type, specifying whether the file is a directory, append-only file, etc., and two unsigned integers: first the four-byte qid version, then the eight-byte qid path. The path is an integer unique among all files in the hierarchy. If a file is deleted and recreated with the same name in the same directory, the old and new path components of the qids should be different. The version is a version number for a file; typically, it is incremented every time the file is modified.

An existing file can be opened, or a new file may be created in the current (directory) file. I/O of a given number of bytes at a given offset on an open file is done by read and write.

A client should cluster any fid that is no longer needed. The remove transaction deletes files.

The stat transaction retrieves information about the file. The stat field in the reply includes the file’s name, access permissions (read, write and execute for owner, group and public), access and modification times, and owner and group identifications (see stat(2)). The owner and group identifications are textual names. The wstat transaction allows some of a file’s properties to be changed.

A request can be aborted with a flush request. When a server receives a Rflush, it should not reply to the message with tag oldtag (unless it has already replied), and it should immediately send an RFflush. The client must wait until it gets the Rflush (even if the reply to the original message arrives in the interim), at which point oldtag may be reused.

Because the message size is negotiable and some elements of the protocol are variable length, it is possible (although unlikely) to have a situation where a valid message is too large to fit within the negotiated size. For example, a very long file name may cause a Rstat of the file or Rread of its directory entry to be too large to send. In most such cases, the server should generate an error rather than modify the data to fit, such as by truncating the file name. The exception is that a long error string in an Rerror message should be truncated if necessary, since the string is only advisory and in some sense arbitrary.

Most programs do not see the 9P protocol directly; instead calls to library routines that access files are translated by the mount driver, mnt(3), into 9P messages.

DIRECTORIES
Directories are created by create with DMDIR set in the permissions argument (see stat(5)). The members of a directory can be found with read(5). All directories must support walks to the directory . . (dot-dot) meaning parent directory, although by convention directories contain no explicit entry for . . or . (dot). The parent of the root directory of a server’s tree is itself.

ACCESS PERMISSIONS
Each file server maintains a set of user and group names. Each user can be a member of any number of groups. Each group has a group leader who has special privileges (see stat(5) and users(6)). Every file request has an implicit user id (copied from the original attach) and an implicit set of groups (where the user is a member).

Each file has an associated owner and group id and three sets of permissions: those of the owner, those of the group, and those of “other” users. When the owner attempts to do something to a file, the owner, group, and other permissions are consulted, and if any of them grant the requested permission, the operation is allowed. For someone who is not the owner, but is a member of the file’s group, the group and other permissions are consulted. For everyone else, the other permissions are used. Each set of permissions says whether reading is allowed, whether writing is allowed, and whether executing is allowed. A walk in a directory is regarded as executing the directory, not reading it. Permissions are kept in the low-order bits of the file mode: owner read/write/execute permission represented as 1 in bits 8, 7, and 6 respectively (using 0 to number the low order). The group permissions are in bits 5, 4, and 3, and the other permissions are in bits 2, 1, and 0.

The file mode contains some additional attributes besides the permissions. If bit 31 (DMDIR) is set, the file is a directory; if bit 30 (DMAPPEND) is set, the file is append-only (offset is ignored in writes); if bit 29 (DMEXCL) is set, the file is exclusive-use (only one client may have it open at a
time); if bit 27 (DMAUTH) is set, the file is an authentication file established by auth messages; if bit 26 (DMTMP) is set, the contents of the file (or directory) are not included in nightly archives. (Bit 28 is skipped for historical reasons.) These bits are reproduced, from the top bit down, in the type byte of the Qid: QTDIR, QTAPPEND, QTEXCL, (skipping one bit) QTAUTH, and QTMP. The name QTFILE, defined to be zero, identifies the value of the type for a plain file.
NAME
attach, auth - messages to establish a connection

SYNOPSIS

DESCRIPTION
The attach message serves as a fresh introduction from a user on the client machine to the
server. The message identifies the user (uname) and may select the file tree to access (aname).
The afid argument specifies a fid previously established by an auth message, as described below.

As a result of the attach transaction, the client will have a connection to the root directory of the
desired file tree, represented by fid. An error is returned if fid is already in use. The server's idea
of the root of the file tree is represented by the returned qid.

If the client does not wish to authenticate the connection, or knows that authentication is not
required, the afid field in the attach message should be set to NOFID, defined as
(u32int)-0 in <fcall.h>. If the client does wish to authenticate, it must acquire and validate an afid using
an auth message before doing the attach.

The auth message contains afid, a new fid to be established for authentication, and the uname
and aname that will be those of the following attach message. If the server does not require
authentication, it returns Rerror to the Tauth message.

If the server does require authentication, it returns aqid defining a file of type QTAUTH (see
intro(5)) that may be read and written (using read and write messages in the usual way) to execute
an authentication protocol. That protocol's definition is not part of 9P itself.

Once the protocol is complete, the same afid is presented in the attach message for the user,
granting entry. The same validated afid may be used for multiple attach messages with the
same uname and aname.

ENTRY POINTS
An attach transaction will be generated for kernel devices (see intro(5)) when a system call eval-
uates a file name beginning with #. Pipe(2) generates an attach on the kernel device pipe(3). The
mount system call (see bind(2)) generates an attach message to the remote file server. When the
kernel boots, an attach is made to the root device, root(3), and then an attach is made to the
requested file server machine.

An auth transaction is generated by the fauth(2) system call or by the first mount system call on
an uninitialized connection.

SEE ALSO
auth(2), fauth(2), version(5), authsrv(6)
NAME
cluck – forget about a fid

SYNOPSIS

DESCRIPTION
The clunk request informs the file server that the current file represented by fid is no longer needed by the client. The actual file is not removed on the server unless the fid had been opened with ORCLOSE.

Once a fid has been clunked, the same fid can be reused in a new walk or attach request.
Even if the clunk returns an error, the fid is no longer valid.

ENTRY POINTS
A clunk message is generated by close and indirectly by other actions such as failed open calls.
NAME
  error - return an error

SYNOPSIS

DESCRIPTION
  The Rerror message (there is no Terror) is used to return an error string describing the failure of a transaction. It replaces the corresponding reply message that would accompany a successful call; its tag is that of the failing request.

  By convention, clients may truncate error messages after ERRMAX–1 bytes; ERRMAX is defined in <libc.h>.
NAME
flush - abort a message

SYNOPSIS

DESCRIPTION
When the response to a request is no longer needed, such as when a user interrupts a process doing a read(2), a Tflush request is sent to the server to purge the pending response. The message being flushed is identified by oldtag. The semantics of flush depends on messages arriving in order.

The server should answer the flush message immediately. If it recognizes oldtag as the tag of a pending transaction, it should abort any pending response and discard that tag. In either case, it should respond with an Rflush echoing the tag (not oldtag) of the Tflush message. A Tflush can never be responded to by an Rerror message.

The server may respond to the pending request before responding to the Tflush. It is possible for a client to send multiple Tflush messages for a particular pending request. Each subsequent Tflush must contain as oldtag the tag of the pending request (not a previous Tflush). Should multiple Tflushes be received for a pending request, they must be answered in order. A Rflush for any of the multiple Tflushes implies an answer for all previous ones. Therefore, should a server receive a request and then multiple flushes for that request, it need respond only to the last flush.

When the client sends a Tflush, it must wait to receive the corresponding Rflush before reusing oldtag for subsequent messages. If a response to the flushed request is received before the Rflush, the client must honor the response as if it had not been flushed, since the completed request may signify a state change in the server. For instance, Tcreate may have created a file and Twalk may have allocated a fid. If no response is received before the Rflush, the flushed transaction is considered to have been canceled, and should be treated as though it had never been sent.

Several exceptional conditions are handled correctly by the above specification: sending multiple flushes for a single tag, flushing after a transaction is completed, flushing a Tflush, and flushing an invalid tag.
NAME
open, create - prepare a fid for I/O on an existing or new file

SYNOPSIS

DESCRIPTION
The open request asks the file server to check permissions and prepare a fid for I/O with subsequent read and write messages. The mode field determines the type of I/O: 0 (called OREAD in <libc.h>), 1 (WRITE), 2 (ORDWR), and 3 (OEXEC) mean read access, write access, read and write access, and execute access, to be checked against the permissions for the file. In addition, if mode has the OTRUNC (0x10) bit set, the file is to be truncated, which requires write permission (if the file is append-only, and permission is granted, the open succeeds but the file will not be truncated); if the mode has the OTRUNC (0x10) bit set, the file is to be removed when the fid is clunked, which requires permission to remove the file from its directory. All other bits in mode should be zero. It is illegal to write a directory, truncate it, or attempt to remove it on close. If the file is marked for exclusive use (see stat(5)), only one client can have the file open at any time. That is, after such a file has been opened, further opens will fail until fid has been clunked. All these permissions are checked at the time of the open request; subsequent changes to the permissions of files do not affect the ability to read, write, or remove an open file.

The create request asks the file server to create a new file with the name supplied, in the directory (dir) represented by fid, and requires write permission in the directory. The owner of the file is the implied user id of the request, the group of the file is the same as dir, and the permissions are the value of

 perm & (~0666 | (dir.perm & 0666))
if a regular file is being created and
 perm & (~0777 | (dir.perm & 0777))
if a directory is being created. This means, for example, that if the create allows read permission to others, but the containing directory does not, then the created file will not allow others to read the file.

Finally, the newly created file is opened according to mode, and fid will represent the newly opened file. Mode is not checked against the permissions in perm. The qid for the new file is returned with the create reply message.

Directories are created by setting the DMDIR bit (0x80000000) in the perm.

The names . and .. are special; it is illegal to create files with these names.

It is an error for either of these messages if the fid is already the product of a successful open or create message.

An attempt to create a file in a directory where the given name already exists will be rejected; in this case, the create system call (see open(2)) uses open with truncation. The algorithm used by the create system call is: first walk to the directory to contain the file. If that fails, return an error. Next walk to the specified file. If the walk succeeds, send a request to open and truncate the file and return the result, successful or not. If the walk fails, send a create message. If that fails, it may be because the file was created by another process after the previous walk failed, so (once) try the walk and open again.

For the behavior of create on a union directory, see bind(2).

The iounit field returned by open and create may be zero. If it is not, it is the maximum number of bytes that are guaranteed to be read from or written to the file without breaking the I/O transfer into multiple 9P messages; see read(5).

ENTRY POINTS
Open and create both generate open messages; only create generates a create message. The iounit associated with an open file may be discovered by calling iounit(2).
For programs that need atomic file creation, without the race that exists in the open-create sequence described above, the kernel does the following. If the OEXCL (0x1000) bit is set in the mode for a create system call, the open message is not sent; the kernel issues only the create. Thus, if the file exists, create will draw an error, but if it doesn't and the create system call succeeds, the process issuing the create is guaranteed to be the one that created the file.
NAME
read, write - transfer data from and to a file

SYNOPSIS

DESCRIPTION
The read request asks for count bytes of data from the file identified by fid, which must be
opened for reading, starting offset bytes after the beginning of the file. The bytes are returned
with the read reply message.

The count field in the reply indicates the number of bytes returned. This may be less than the
requested amount. If the offset field is greater than or equal to the number of bytes in the file, a
count of zero will be returned.

For directories, read returns an integral number of directory entries exactly as in stat (see
stat(5)), one for each member of the directory. The read request message must have offset
equal to zero or the value of offset in the previous read on the directory, plus the number of
bytes returned in the previous read. In other words, seeking other than to the beginning is illegal
in a directory (see seek(2)).

The write request asks that count bytes of data be recorded in the file identified by fid, which
must be opened for writing, starting offset bytes after the beginning of the file. If the file is
append-only, the data will be placed at the end of the file regardless of offset. Directories may not
be written.

The write reply records the number of bytes actually written. It is usually an error if this is not
the same as requested.

Because 9P implementations may limit the size of individual messages, more than one message
may be produced by a single read or write call. The iounit field returned by open(5), if non-zero,
reports the maximum size that is guaranteed to be transferred atomically.

ENTRY POINTS
Read and write messages are generated by the corresponding calls. Because they include an
offset, the pread and pwrite calls correspond more directly to the 9P messages. Although seek(2)
affects the offset, it does not generate a message.
NAME
remove - remove a file from a server

SYNOPSIS

DESCRIPTION
The remove request asks the file server both to remove the file represented by fid and to clunk the fid, even if the remove fails. This request will fail if the client does not have write permission in the parent directory.

It is correct to consider remove to be a clunk with the side effect of removing the file if permissions allow.

If a file has been opened as multiple fids, possibly on different connections, and one fid is used to remove the file, whether the other fids continue to provide access to the file is implementation-defined. The Plan 9 file servers (like fs(4)) remove the file immediately: attempts to use the other fids will yield a "phase error." U9fs(4) follows the semantics of the underlying Unix file system, so other fids typically remain usable.

ENTRY POINTS
Remove messages are generated by remove.
NAME
stat, wstat – inquire or change file attributes

SYNOPSIS

DESCRIPTION
The stat transaction inquires about the file identified by fid. The reply will contain a machine-

independent directory entry, stat, laid out as follows:
size[2] total byte count of the following data

  type[2]
    for kernel use
dev[4] for kernel use
qid.type[1]
  the type of the file (directory, etc.), represented as a bit vector corresponding to the high 8
  bits of the file’s mode word.
qid.vers[4]
  version number for given path
qid.path[8]
  the file server’s unique identification for the file
mode[4]
  permissions and flags
atime[4]
  last access time
mtime[4]
  last modification time
length[8]
  length of file in bytes
name[ s ]
  file name; must be / if the file is the root directory of the server
uid[ s ]
  owner name
gid[ s ]
  group name
muid[ s ]
  name of the user who last modified the file

Integers in this encoding are in little-endian order (least significant byte first). The convM2D and
convD2M routines (see fcall(2)) convert between directory entries and a C structure called a Dir.

The mode contains permission bits as described in intro(5) and the following: 0x80000000
(DMDIR, this file is a directory), 0x40000000 (DMAPPEND, append only), 0x20000000
(DMEXCL, exclusive use), 0x04000000 (DMIMP, temporary); these are echoed in Qid.type.
Writes to append-only files always place their data at the end of the file; the offset in the write
message is ignored, as is the OTRUNC bit in an open. Exclusive use files may be open for I/O by
only one fid at a time across all clients of the server. If a second open is attempted, it draws an
error. Servers may implement a timeout on the lock on an exclusive use file: if the fid holding the
file open has been unused for an extended period (of order at least minutes), it is reasonable to
break the lock and deny the initial fid further I/O. Temporary files are not included in nightly
archives (see fossil(4)).
The two time fields are measured in seconds since the epoch (Jan 1 00:00 1970 GMT). The mtime field reflects the time of the last change of content (except when later changed by wstat). For a plain file, mtime is the time of the most recent create, open with truncation, or write; for a directory it is the time of the most recent remove, create, or wstat of a file in the directory. Similarly, the atime field records the last read of the contents; also it is set whenever mtime is set. In addition, for a directory, it is set by an attach, walk, or create, all whether successful or not.

The muid field names the user whose actions most recently changed the mtime of the file.

The length records the number of bytes in the file. Directories and most files representing devices have a conventional length of 0.

The stat request requires no special permissions.

The wstat request can change some of the file status information. The name can be changed by anyone with write permission in the parent directory; it is an error to change the name to that of an existing file. The length can be changed (affecting the actual length of the file) by anyone with write permission on the file. It is an error to attempt to set the length of a directory to a non-zero value, and servers may decide to reject length changes for other reasons. The mode and mtime can be changed by the owner of the file or the group leader of the file’s current group. The directory bit cannot be changed by a wstat; the other defined permission and mode bits can. The gid can be changed: by the owner if also a member of the new group; or by the group leader of the file’s current group if also leader of the new group (see intro(5) for more information about permissions and users(6) for users and groups). None of the other data can be altered by a wstat: attempts to change them will trigger an error. In particular, it is illegal to attempt to change the owner of a file. (These conditions may be relaxed when establishing the initial state of a file server; see fsconfig(8).)

Either all the changes in wstat request happen, or none of them does: if the request succeeds, all changes were made; if it fails, none were.

A wstat request can avoid modifying some properties of the file by providing explicit “don’t touch” values in the stat data that is sent: zero-length strings for text values and the maximum unsigned value of appropriate size for integral values. As a special case, if all the elements of the directory entry in a Twstat message are “don’t touch” values, the server may interpret it as a request to guarantee that the contents of the associated file are committed to stable storage before the Rwstat message is returned. (Consider the message to mean, “make the state of the file exactly what it claims to be.”)

A read of a directory yields an integral number of directory entries in the machine independent encoding given above (see read(5)).

Note that since the stat information is sent as a 9P variable–length datum, it is limited to a maximum of 65535 bytes.

ENTRY POINTS

Stat messages are generated by fstat and stat.
Wstat messages are generated by fwstat and wstat.

BUGS

To make the contents of a directory, such as returned by read(5), easy to parse, each directory entry begins with a size field. For consistency, the entries in Twstat and Rstat messages also contain their size, which means the size appears twice. For example, the Rstat message is formatted as “(4+1+2+2+n)[4] Rstat tag[2] n[2] (n-2)[2] type[2] dev[4]...,” where n is the value returned by convD2M.
NAME
   version - negotiate protocol version

SYNOPSIS

DESCRIPTION
   The version request negotiates the protocol version and message size to be used on the connection and initializes the connection for I/O. Tversion must be the first message sent on the 9P connection, and the client cannot issue any further requests until it has received the Rversion reply. The tag should be NOTAG (value (ushort)~0) for a version message.

   The client suggests a maximum message size, msize, that is the maximum length, in bytes, it will ever generate or expect to receive in a single 9P message. This count includes all 9P protocol data, starting from the size field and extending through the message, but excludes enveloping transport protocols. The server responds with its own maximum, msize, which must be less than or equal to the client's value. Thenceforth, both sides of the connection must honor this limit.

   The version string identifies the level of the protocol. The string must always begin with the two characters “9P”. If the server does not understand the client's version string, it should respond with an Rversion message (not Rerror) with the version string the 7 characters “unknown”.

   The server may respond with the client's version string, or a version string identifying an earlier defined protocol version. Currently, the only defined version is the 6 characters “9P2000”. Version strings are defined such that, if the client string contains one or more period characters, the initial substring up to but not including any single period in the version string defines a version of the protocol. After stripping any such period-separated suffix, the server is allowed to respond with a string of the form 9Pnnnn, where nnnn is less than or equal to the digits sent by the client.

   The client and server will use the protocol version defined by the server's response for all subsequent communication on the connection.

   A successful version request initializes the connection. All outstanding I/O on the connection is aborted; all active fids are freed ('clunked') automatically. The set of messages between version requests is called a session.

ENTRY POINTS
   The version message is generated by the VERSION system call. It is also generated automatically, if required, by a mount or fauth system call on an uninitialized connection.
NAME
walk - descend a directory hierarchy

SYNOPSIS

DESCRIPTION
The walk request carries as arguments an existing fid and a proposed newfid (which must not be
in use unless it is the same as fid) that the client wishes to associate with the result of traversing
the directory hierarchy by ‘walking’ the hierarchy using the successive path name elements
wname. The fid must represent a directory unless zero path name elements are specified.

The fid must be valid in the current session and must not have been opened for I/O by an open or
create message. If the full sequence of nwname elements is walked successfully, newfid will
represent the file that results. If not, newfid and fid will be unaffected. However, if newfid is in
use or otherwise illegal, an Rerror is returned.

The name “..” (dot-dot) represents the parent directory. The name “.” (dot), meaning the cur-
rent directory, is not used in the protocol.

It is legal for nwname to be zero, in which case newfid will represent the same file as fid and the
walk will usually succeed; this is equivalent to walking to dot. The rest of this discussion
assumes nwname is greater than zero.

The nwname path name elements wname are walked in order, “elementwise”. For the first ele-
mentwise walk to succeed, the file identified by fid must be a directory, and the implied user of the
request must have permission to search the directory (see intro(5)). Subsequent elementwise walks
have equivalent restrictions applied to the implicit fid that results from the preceding elementwise
walk.

If the first element cannot be walked for any reason, Rerror is returned. Otherwise, the walk will
return an Rwalk message containing nwqid qids corresponding, in order, to the files that are vis-
ited by the nwqid successful elementwise walks; nwqid is therefore either nwname or the index of
the first elementwise walk that failed. The value of nwqid cannot be zero unless nwname is zero.
Also, nwqid will always be less than or equal to nwname. Only if it is equal, however, will newfid
be affected, in which case newfid will represent the file reached by the final elementwise walk
requested in the message.

A walk of the name “..” in the root directory of a server is equivalent to a walk with no name
elements.

If newfid is the same as fid, the above discussion applies, with the obvious difference that if the
walk changes the state of newfid, it also changes the state of fid; and if newfid is unaffected, then
fid is also unaffected.

To simplify the implementation of the servers, a maximum of sixteen name elements or qids may
be packed in a single message. This constant is called MAXWELEM in fcall(2). Despite this restric-
tion, the system imposes no limit on the number of elements in a file name, only the number that
may be transmitted in a single message.

ENTRY POINTS
A call to chdir(2) causes a walk. One or more walk messages may be generated by any of the
following calls, which evaluate file names: bind, create, exec, mount, open, remove, stat,
unmount, wstat. The file name element . (dot) is interpreted locally and is not transmitted in
walk messages.
NAME
intro - introduction to file formats

DESCRIPTION
This section of the manual describes file formats and other miscellany such as troff macro packages.
NAME
a.out - object file format

SYNOPSIS
#include <a.out.h>

DESCRIPTION
An executable Plan 9 binary file has up to six sections: a header, the program text, the data, a symbol table, a PC/SP offset table (MC68020 only), and finally a PC/line number table. The header, given by a structure in <a.out.h>, contains 4-byte integers in big-endian order:

typedef struct Exec {
    long magic;    /* magic number */
    long text;     /* size of text segment */
    long data;     /* size of initialized data */
    long bss;      /* size of uninitialized data */
    long syms;     /* size of symbol table */
    long entry;    /* entry point */
    long spsz;     /* size of pc/sp offset table */
    long pcsz;     /* size of pc=line number table */
} Exec;
#define _MAGIC(b) ((((4*b)+0)*b)+7)
#define A_MAGIC _MAGIC(8) /* 68020 */
#define I_MAGIC _MAGIC(11) /* intel 386 */
#define J_MAGIC _MAGIC(12) /* intel 960 */
#define K_MAGIC _MAGIC(13) /* sparc */
#define V_MAGIC _MAGIC(16) /* mips 3000 */
#define X_MAGIC _MAGIC(17) /* att dsp 3210 */
#define M_MAGIC _MAGIC(18) /* mips 4000 */
#define D_MAGIC _MAGIC(19) /* amd 29000 */
#define E_MAGIC _MAGIC(20) /* arm 7-something */
#define Q_MAGIC _MAGIC(21) /* powerpc */
#define N_MAGIC _MAGIC(22) /* mips 4000 LE */
#define L_MAGIC _MAGIC(23) /* dec alpha */

Sizes are expressed in bytes. The size of the header is not included in any of the other sizes.

When a Plan 9 binary file is executed, a memory image of three segments is set up: the text segment, the data segment, and the stack. The text segment begins at a virtual address which is a multiple of the machine-dependent page size. The text segment consists of the header and the first text bytes of the binary file. The entry field gives the virtual address of the entry point of the program. The data segment starts at the first page-rounded virtual address after the text segment. It consists of the next data bytes of the binary file, followed by bss bytes initialized to zero. The stack occupies the highest possible locations in the core image, automatically growing downwards. The bss segment may be extended by brk(2).

The next syms (possibly zero) bytes of the file contain symbol table entries, each laid out as:

    uchar value[4];
    char type;
    char name[\n];    /* NUL-terminated */

The value is in big-endian order and the size of the name field is not pre-defined: it is a zero-terminated array of variable length.

The type field is one of the following characters with the high bit set:

    T  text segment symbol
    t  static text segment symbol
    L  leaf function text segment symbol
    l  static leaf function text segment symbol
    D  data segment symbol
    d  static data segment symbol

752
B  bss segment symbol
b  static bss segment symbol
a  automatic (local) variable symbol
p  function parameter symbol

A few others are described below. The symbols in the symbol table appear in the same order as the program components they describe.

The Plan 9 compilers implement a virtual stack frame pointer rather than dedicating a register; moreover, on the MC680X0 architectures there is a variable offset between the stack pointer and the frame pointer. Following the symbol table, MC680X0 executable files contain a spsz–byte table encoding the offset of the stack frame pointer as a function of program location; this section is not present for other architectures. The PC/SP table is encoded as a byte stream. By setting the PC to the base of the text segment and the offset to zero and interpreting the stream, the offset can be computed for any PC. A byte value of 0 is followed by four bytes that hold, in big–endian order, a constant to be added to the offset. A byte value of 1 to 64 is multiplied by four and added, without sign extension, to the offset. A byte value of 65 to 128 is reduced by 64, multiplied by four, and subtracted from the offset. A byte value of 129 to 255 is reduced by 129, multiplied by the quantum of instruction size (e.g. two on the MC680X0), and added to the current PC without changing the offset. After any of these operations, the instruction quantum is added to the PC.

A similar table, occupying pc sz–bytes, is the next section in an executable; it is present for all architectures. The same algorithm may be run using this table to recover the absolute source line number from a given program location. The absolute line number (starting from zero) counts the newlines in the C–preprocessed source seen by the compiler. Three symbol types in the main symbol table facilitate conversion of the absolute number to source file and line number:

  f  source file name components
  z  source file name
  Z  source file line offset

The f symbol associates an integer (the value field of the ‘symbol’) with a unique file path name component (the name of the ‘symbol’). These path components are used by the z symbol to represent a file name: the first byte of the name field is always 0; the remaining bytes hold a zero–terminated array of 16–bit values (in big–endian order) that represent file name components from f symbols. These components, when separated by slashes, form a file name. The initial slash of a file name is recorded in the symbol table by an f symbol; when forming file names from z symbols an initial slash is not to be assumed. The z symbols are clustered, one set for each object file in the program, before any text symbols from that object file. The set of z symbols for an object file form a history stack of the included source files from which the object file was compiled. The value associated with each z symbol is the absolute line number at which that file was included in the source; if the name associated with the z symbol is null, the symbol represents the end of an included file, that is, a pop of the history stack. If the value of the z symbol is 1 (one), it represents the start of a new history stack. To recover the source file and line number for a program location, find the text symbol containing the location and then the first history stack preceding the text symbol in the symbol table. Next, interpret the PC/line offset table to discover the absolute line number for the program location. Using the line number, scan the history stack to find the set of source files open at that location. The line number within the file can be found using the line numbers in the history stack. The Z symbols correspond to #line directives in the source; they specify an adjustment to the line number to be printed by the above algorithm. The offset is associated with the first previous z symbol in the symbol table.

SEE ALSO
db(1), acid(1), 2a(1), 2l(1), nm(1), strip(1), mach(2), symbol(2)

BUGS
There is no type information in the symbol table; however, the –a flags on the compilers will produce symbols for acid(1).
NAME
ar - archive (library) file format

SYNOPSIS
#include <ar.h>

DESCRIPTION
The archive command ar(1) is used to combine several files into one. Archives are used mainly as libraries to be searched by the loaders 2l(1) et al.

A file produced by ar has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

#define ARMAG "!\n"
#define SARMAG 8
#define ARFMAG "\n"

struct ar_hdr {
    char name[16];
    char date[12];
    char uid[6];
    char gid[6];
    char mode[8];
    char size[10];
    char fmag[2];
};
#define SAR_HDR 60

The name is a blank–padded string. The fmag field contains ARFMAG to help verify the presence of a header. The other fields are left–adjusted, blank–padded numbers. They are decimal except for mode, which is octal. The date is the modification date of the file (see stat(2)) at the time of its insertion into the archive. The mode is the low 9 bits of the file permission mode. The length of the header is SAR_HDR. Because the ar_hdr structure is padded in an architecture–dependent manner, the structure should never be read or written as a unit; instead, each field should be read or written independently.

Each file begins on an even (0 mod 2) boundary; a newline is inserted between files if necessary. Nevertheless size reflects the actual size of the file exclusive of padding.

When all members of an archive are object files of the same architecture, ar automatically adds an extra file, named __.SYMDEF, as the first member of the archive. This file contains an index used by the loaders to locate all externally defined text and data symbols in the archive.

There is no provision for empty areas in an archive file.

SEE ALSO
ar(1), 2l(1), nm(1), stat(2)

BUGS
The uid and gid fields are unused in Plan 9. They provide compatibility with Unix ar format.
NAME
authsrv, p9any, p9sk1, p9sk2 - authentication protocols

DESCRIPTION
This manual page describes the protocols used to authorize connections, confirm the identities of users and machines, and maintain the associated databases. The machine that provides these services is called the authentication server (AS). The AS may be a stand-alone machine or a general-use machine such as a CPU server. The network database ndb(6) holds for each public machine, such as a CPU server or file server, the name of the authentication server that machine uses.

Each machine contains three values important to authentication; a 56-bit DES key, a 28-byte authentication ID, and a 48-byte authentication domain name. The ID is a user name and identifies who is currently responsible for the kernel running on that machine. The domain name identifies the machines across which the ID is valid. Together, the ID and domain name identify the owner of a key.

When a terminal boots, factotum(4) prompts for user name and password. The user name becomes the terminal’s authentication ID. The password is converted using passtokey (see authsrv(2)) into a 56-bit DES key and saved in memory. The authentication domain is set to the null string. If possible, factotum validates the key with the AS before saving it. For Internet machines the correct AS to ask is found using dhcpd(8).

When a CPU or file server boots, factotum reads the key, ID, and domain name from non-volatile RAM. This allows servers to reboot without operator intervention.

The details of any authentication are mixed with the semantics of the particular service they are authenticating so we describe them one case at a time. The following definitions will be used in the descriptions:

- $K_s$: server’s host ID’s key
- $K_c$: client’s host ID’s key
- $K_n$: a nonce key created for a ticket (key)
- $K[m]$: message $m$ encrypted with key $K$
- $C H_c$: an 8-byte random challenge from a client (chal)
- $C H_s$: an 8-byte random challenge from a server (chal)
- $I D_s$: server’s ID (authid)
- $D N$: server’s authentication domain name (authdom)
- $I D_c$: client’s ID (hostid, cuid)
- $I D_r$: client’s desired ID on server (uid, cuid)

The parenthesized names are the ones used in the Ticketreq and Ticket structures in <authsrv.h>.

The message type constants AuthTreq, AuthChal, AuthPass, AuthOK, AuthErr, AuthMod, AuthApop, AuthOKvar, Auth Chap, AuthMSchap, AuthCram, and AuthVNC (type) are defined in <authsrv.h>, as are the encrypted message types AuthTs, AuthAs, AuthAc, AuthTp, and AuthHr (num).

Ticket Service
When a client and server wish to authenticate to each other, they do so using tickets issued by the AS. Obtaining tickets from the AS is the client’s responsibility.

The protocol to obtain a ticket pair is:

- $C \rightarrow A \quad \text{AuthTreq, } I D_s, \text{ DN, } C H_s, \text{ IDc, } I D_r$
- $A \rightarrow C \quad \text{AuthOK, } K_c(\text{AuthTc, } C H_s, \text{ IDc, } I D_r, \text{ Kn}), K_s(\text{AuthTs, } C H_s, \text{ IDc, } I D_r, \text{ Kn})$

The two tickets are identical except for their type fields and the keys with which they are encrypted. The client and server can each decrypt one of the tickets, establishing a shared secret Kn.

The tickets can be viewed as a statement by the AS that “a client possessing the Kn key is allowed to authenticate as IDr.”

The presence of the server challenge CHs in the ticket allows the server to verify the freshness of the ticket pair.
The AS sets the IDr in the tickets to the requested IDr only if IDC is allowed to speak for (q.v.) IDr. If not, the AS sets IDr to the empty string.

If the users IDC or IDs do not exist, the AS silently generates one-time random keys to use in place of Kc or Ks, so that clients cannot probe the AS to learn whether a user name is valid.

P9sk1
The Plan 9 shared key protocol p9sk1 allows a client and server to authenticate each other. The protocol is:

\texttt{C \rightarrow S \ CHc}

The client starts by sending a random challenge to the server.

\texttt{S \rightarrow C \ AuthTreq, IDs, DN, CHs, \ldots}

The server replies with a ticket request giving its id and authentication domain along with its own random challenge.

\texttt{C \rightarrow S \ Ks[AuthTs, CHs, IDC, IDr, Kc], Kn[AuthAc, CHs]}

The client adds IDC and IDr to the ticket request and obtains a ticket pair from the AS as described above. The client relays the server’s ticket along with an authenticator, the AuthAc message. The authenticator proves to the server that the client knows Kc and is therefore allowed to authenticate as IDr. (The inclusion of CHs in the authenticator avoids replay attacks.)

\texttt{S \rightarrow C \ Kn[AuthAs, CHc]}

The server replies with its own authenticator, proving to the client that it also knows Kc and therefore Ks.

P9sk2 is an older variant of p9sk1 used only when connecting to pre–9P2000 remote execution services. It omits the first message and last messages and therefore does not authenticate the server to the client.

P9any

P9any is the standard Plan 9 authentication protocol. It consists of a negotiation to determine a common protocol, followed by the agreed-upon protocol.

The negotiation protocol is:

\texttt{S \rightarrow C \ v.2 proto@authdom proto@authdom ...}

\texttt{C \rightarrow S \ proto dom}

\texttt{S \rightarrow C \ OK}

Each message is a NUL-terminated UTF string. The server begins by sending a list of proto, authdom pairs it is willing to use. The client responds with its choice. Requiring the client to wait for the final OK ensures that the client will not start the chosen protocol until the server is ready.

The above is version 2 of the protocol. Version 1, no longer used, omitted the first message’s v.2 prefix and the OK message.

The p9any protocol is the protocol used by all Plan 9 services. The file server runs it over special authentication files (see auth(2) and attach(5)). Other services, such as cpu(1) and exportfs(4), run p9any over the network and then use Kn to derive an ssl(3) key to encrypt the rest of their communications.

Password Change

Users connect directly to the AS to change their passwords. The protocol is:

\texttt{C \rightarrow A \ AuthPass, IDC, DN, CHc, IDC, IDC}

The client sends a password change ticket request.

\texttt{A \rightarrow C \ Kc[AuthTp, CHc, IDC, IDC, Kc]}

The server responds with a ticket containing the key Kn encrypted with the client’s key Kc.

\texttt{C \rightarrow A \ Kn[AuthPass, old, new, changesecret, secret]}

The client decrypts the ticket using the old password and then sends back an encrypted password request (PasswordReq structure) containing the old password and the new password. If changesecret is set, the AS also changes the user’s secret, the password used for non–Plan 9 authentications.
\textbf{A→C}  
AuthOK or AuthErr, 64-byte error message
The AS responds with simply AuthOK or with AuthErr followed by a 64-byte error message.

\textbf{Authentication Database}
An ndb(2) database file /\texttt{lib/ndb/auth} exists for the AS. This database maintains "speaks for" relationships, i.e., it lists which users may speak to other users when authenticating. The attribute types used by the AS are hostid and uid. The value in the hostid is a client host's ID. The values in the uid pairs in the same entry list which users that host ID make speak for. A uid value of * means the host ID may speak for all users. A uid value of ! user means the host ID may not speak for user. For example:

\begin{verbatim}
hostid=boote
uid=!sys uid=!adm uid=* 
\end{verbatim}

is interpreted as boote may speak for any user except sys and adm. This property is used heavily on CPU servers.

\textbf{Foreign Protocols}
The AS accepts ticket request messages of types other than AuthTreq to allow users to authenticate using non-Plan 9 protocols. In these situations, the server communicates directly with the AS. Some protocols must begin without knowing the client’s name. They ignore the client name in the ticket request. All the protocols end with the AS sending an AuthOK message containing a server ticket and authenticator.

AuthOK messages always have a fixed but context-dependent size. The occasional variable-length OK message starts with a AuthOKvar byte and a five-byte space-padded decimal length of the data that follows.

Anywhere an AuthOK message is expected, a AuthErr message may be substituted.

\textbf{S→A}  \texttt{AuthChal, IDs, DN, CHs, IDc, ID}\texttt{d}
\textbf{A→S}  AuthOK, \texttt{challenge}
\textbf{S→A}  \texttt{response}
\textbf{A→S}  AuthOK, Ks\{AuthChal, IDs, DN, CHs, IDc, IDc, Kn\}, Kn\{AuthTs, CHs\}

This protocol allows the use of handheld authenticators such as SecureNet keys and SecureID tokens in programs such as ssh(1) and ftpd (see ipserv(8)).

\texttt{Challenge} and \texttt{response} are text strings, NUL-padded to 16 bytes (NETCHLEN). The \texttt{challenge} is a random five-digit decimal number. When using a SecureNet key or netkey (see passwd(1)), the \texttt{response} is an eight-digit decimal or hexadecimal number that is an encryption of the challenge using the user's DES key.

When using a SecureID token, the challenge is ignored. The response is the user’s PIN followed by the six-digit number currently displayed on the token. In this case, the AS queries an external RADIUS server to check the response. Use of a RADIUS server requires an entry in the authentication database. For example:

\begin{verbatim}
radius=server-name secret=xyzzy
uid=howard rid=trickey
uid=sape rid=smullender
\end{verbatim}

In this example, the secret xyzzy is the hash key used in talking to the RADIUS server. The uid/rid lines map from Plan 9 user ids to RADIUS ids. Users not listed are assumed to have the same id in both places.

\textbf{S→A}  \texttt{AuthApop, IDs, DN, CHs, \texttt{─,\texttt{─}}}
\textbf{A→S}  AuthOKvar, \texttt{challenge}
\textbf{S→A}  AuthApop, IDs, DN, CHs, IDc, IDc; hexadecimal MD5 checksum
\textbf{A→S}  AuthOK, Ks\{AuthApop, IDs, DN, CHs, IDc, IDc, Kn\}, Kn\{AuthTs, CHs\}

This protocol implements APOP authentication (see pop3(8)). After receiving a ticket request of type AuthApop, the AS generates a random challenge of the form <random@domain>. The client then replies with a new ticket request giving the user name followed by the MD5 checksum of the challenge concatenated with the user's secret. If the response is correct, the authentication server sends back a ticket and authenticator. If the response is incorrect, the client may repeat the ticket request/MD5 checksum message to try again.
The AuthCram protocol runs identically to the AuthApop protocol, except that the expected
MD5 checksum is the keyed MD5 hash using the user’s secret as the key (see hmac_md5 in
sechash(2)).

S→A AuthChap, IDs, DN, CHs, -, -
A→S challenge
S→A pktid, IDc, response
A→S AuthOK, Ks{AuthChap, IDs, DN, CHs, IDs, IDc, Kn}, Kn{AuthTs, CHs}

This protocol implements CHAP authentication (see ppp(8)). The challenge is eight random
bytes. The response is a 16-byte MD5 checksum over the packet id, user’s secret, and
challenge. The reply packet is defined as OChapreply in <authsrv.h>.

S→A AuthMSchap, IDs, DN, CHs, -, -
A→S challenge
S→A IDc, lm−response, nt−response
A→S AuthOK, Ks{AuthMSchap, IDs, DN, CHs, IDs, IDc, Kn}, Kn{AuthTs, CHs}

This protocol implements Microsoft’s MS−CHAP authentication (see ppp(8)). The challenge
is eight random bytes. The two responses are Microsofts LM and NT hashes. Only the NT
hash may be used to authenticate, as the LM hash is considered too weak. The reply
packet is defined as OMSChapreply in <authsrv.h>.

S→A AuthVNC, IDs, DN, CHs, IDs, IDc
A→S AuthOKvar, challenge
S→A response
A→S AuthOK, Ks{, IDs, DN, CHs, IDs, IDc, Kn}, Kn{AuthTs, CHs}

This protocol implements VNC authentication (see vncs in vnc(1)). The challenge is 16 ran-
dom bytes, and the response is a DES ECB encryption of the challenge. The method by
which VNC converts the user’s secret into a DES key is weak, considering only the first eight
bytes of the secret.

FILES
/lib/ndb/auth database file
/lib/ndb/auth.* hash files for /lib/ndb/auth

SEE ALSO
auth(2), fauth(2), cons(3), attach(5), auth(8)
NAME

color - representation of pixels and colors

DESCRIPTION

To address problems of consistency and portability among applications, Plan 9 uses a fixed color map, called rgbv, on 8-bit-per-pixel displays. Although this avoids problems caused by multiplexing color maps between applications, it requires that the color map chosen be suitable for most purposes and usable for all. Other systems that use fixed color maps tend to sample the color cube uniformly, which has advantages—mapping from a (red, green, blue) triple to the color map and back again is easy—but ignores an important property of the human visual system: eyes are much more sensitive to small changes in intensity than to changes in hue. Sampling the color cube uniformly gives a color map with many different hues, but only a few shades of each. Continuous tone images converted into such maps demonstrate conspicuous artifacts.

Rather than dice the color cube into subregions of size 6×6×6 (as in Netscape Navigator) or 8×8×4 (as in previous releases of Plan 9), picking 1 color in each, the rgbv color map uses a 4×4×4 subdivision, with 4 shades in each subcube. The idea is to reduce the color resolution by dicing the color cube into fewer cells, and to use the extra space to increase the intensity resolution. This results in 16 grey shades (4 grey subcubes with 4 samples in each), 13 shades of each primary and secondary color (3 subcubes with 4 samples plus black) and a reasonable selection of colors covering the rest of the color cube. The advantage is better representation of continuous tones.

The following function computes the 256 3-byte entries in the color map:

```c
void
setmaprgbv(uchar cmap[256][3])
{
    uchar *c;
    int r, g, b, v;
    int num, den;
    int i, j;

    for(r=0, i=0; r!=4; r++)
        for(v=0; v!=4; v++, i+=16)
            for(g=0, j=v-r; g!=4; g++)
                for(b=0; b!=4; b++, j++){
                    c = cmap[i+(j&15)];
                    den = r;
                    if(g > den)
                        den = g;
                    if(b > den)
                        den = b;
                    if(den == 0) /* would divide check; pick grey shades */
                    else{
                        num = 17*(4*den+v);
                        c[0] = r*num/den;
                        c[1] = g*num/den;
                        c[2] = b*num/den;
                    }
                }
}
```

There are 4 nested loops to pick the (red,green,blue) coordinates of the subcube, and the value (intensity) within the subcube, indexed by r, g, b, and v, whence the name rgbv. The peculiar order in which the color map is indexed is designed to distribute the grey shades uniformly through the map—the i'th grey shade, 0<=i<16 has index i×17, with black going to 0 and white to 255. Therefore, when a call to draw converts a 1, 2 or 4 bit-per-pixel picture to 8 bits per pixel (which it does by replicating the pixels' bits), the converted pixel values are the appropriate grey shades.
The rgbv map is not gamma-corrected, for two reasons. First, photographic film and television are both normally under-corrected, the former by an accident of physics and the latter by NTSC’s design. Second, we require extra color resolution at low intensities because of the non-linear response and adaptation of the human visual system. Properly gamma-corrected displays with adequate low-intensity resolution pack the high-intensity parts of the color cube with colors whose differences are almost imperceptible. Either reason suggests concentrating the available intensities at the low end of the range.

On ‘true-color’ displays with separate values for the red, green, and blue components of a pixel, the values are chosen so 0 represents no intensity (black) and the maximum value (255 for an 8-bit-per-color display) represents full intensity (e.g., full red). Common display depths are 24 bits per pixel, with 8 bits per color in order red, green, blue, and 16 bits per pixel, with 5 bits of red, 6 bits of green, and 5 bits of blue.

Colors may also be created with an opacity factor called alpha, which is scaled so 0 represents fully transparent and 255 represents opaque color. The alpha is premultiplied into the other channels, as described in the paper by Porter and Duff cited in draw(2). The function setalpha (see allocimage(2)) aids the initialization of color values with non-trivial alpha.

The packing of pixels into bytes and words is odd. For compatibility with VGA frame buffers, the bits within a pixel byte are in big-endian order (leftmost pixel is most significant bits in byte), while bytes within a pixel are packed in little-endian order. Pixels are stored in contiguous bytes. This results in unintuitive pixel formats. For example, for the RGB24 format, the byte ordering is blue, green, red.

To maintain a constant external representation, the draw(3) interface as well as the various graphics libraries represent colors by 32-bit numbers, as described in color(2).

SEE ALSO

color(2), graphics(2), draw(2)
NAME
  face - face files

DESCRIPTION
  The directories /usr/$user/lib/face and /lib/face contain a hierarchy of images of people. In those directories are subdirectories named by the sizes of the corresponding image files: 48x48x1 (48 by 48 pixels, one bit per pixel); 48x48x2 (48 by 48 pixels, two (grey) bits per pixel); 48x48x4 (48 by 48 pixels, four (grey) bits per pixel); 48x48x8 (48 by 48 pixels, eight (color-mapped) bits per pixel); 512x512x8 (512 by 512 pixels, eight (color-mapped) bits per pixel); 512x512x24 (512 by 512 pixels, twenty-four bits per pixel (3 times 8 bits per color)). The large files serve no special purpose: they are stored as images (see image(6)). The small files are the 'icons' displayed by faces and seemail (see faces(1)); for depths less than 4, their format is special.

  One- and two-bit deep icons are stored as text, one line of the file to one scan line of display. Each line is divided into 8-bit, 16-bit, or 32-bit big-endian words, stored as a list of comma-separated hexadecimal C constants, such as:

  0x9200, 0x1bb0, 0x003e,

  This odd format is historical and the programs that read it are somewhat forgiving about blanks and the need for commas.

  The files lib/face/*/dict hold a correspondence between users at machines and face files. The format is

  machine/user directory/file.ver

  The machine is the domain name of the machine sending the message, and user the name of the user sending it, as recorded in /sys/log/mail. The directory is a further subdirectory of (say) /lib/face/48x48x1, named by a single letter corresponding to the first character of the user names. The file is the name of the file, typically but not always the user name, and ver is a number to distinguish different images, for example to distinguish the image for Bill Gates from the image for Bill Joy, both of which might otherwise be called b/bill. For example, Bill Gates might be represented by the line

  microsoft.com/bill b/bill.1

  If multiple entries exist for a user in the various .dict files, faces chooses the highest pixel size less than or equal to that of the display on which it is running.

  Finally, or rather firstly, the file /lib/face/.machinelist contains a list of machine/domain pairs, one per line, to map any of a set of machines to a single domain name to be looked up in the .dict files. The machine name may be a regular expression, so for example the entry

  *.research\.bell-labs\.com astro

  maps any of the machines in Bell Labs Research into the shorthand name astro, which then appears as a domain name in the .dict files.

SEE ALSO
  mail(1), tweak(1), image(6)
NAME
font, subfont - external format for fonts and subfonts

SYNOPSIS
#include <draw.h>

DESCRIPTION
Fonts and subfonts are described in cachefonts(2).

External fonts are described by a plain text file that can be read using openfont. The format of the
text is a header followed by any number of subfont range specifications. The header contains two
numbers: the height and the ascent, both in pixels. The height is the inter-line spacing and the
ascent is the distance from the top of the line to the baseline. These numbers are chosen to display
consistently all the subfonts of the font. A subfont range specification contains two or three
numbers and a file name. The numbers are the inclusive range of characters covered by the subfont,
with an optional starting position within the subfont, and the file name names an external file
suitable for readsusubfont (see graphics(2)). The minimum number of a centered range is mapped to
the specified starting position (default zero) of the corresponding subfont. If the subfont file name
does not begin with a slash, it is taken relative to the directory containing the font file. Each field
must be followed by some white space. Each numeric field may be C-format decimal, octal, or
hexadecimal.

External subfonts are represented in a more rigid format that can be read and written using
readsusubfont and writesusubfont (see subfont(2)). The format for subfont files is: an image containing
character glyphs, followed by a subfont header, followed by character information. The image has
the format for external image files described in image(6). The subfont header has 3 decimal
strings: n, height, and ascent. Each number is right-justified and blank padded in 11 characters,
followed by a blank. The character info consists of n+1 6-byte entries, each giving the
FontChar x (2 bytes, low order byte first), top, bottom, left, and width. The x field of
the last FontChar is used to calculate the image width of the previous character; the other fields
in the last FontChar are irrelevant.

Note that the convention of using the character with value zero (NUL) to represent characters of
zero width (see draw(2)) means that fonts should have, as their zeroth character, one with non-zero
width.

FILES
/lib/font/bit/*  font directories

SEE ALSO
graphics(2), draw(2), cachefonts(2), subfont(2)
NAME
htmlroff - HTML formatting and typesetting

DESCRIPTION
Htmlroff(1) accepts troff input with a few extensions and changes. This manual describes the changes to the input language, assuming a working knowledge of troff itself.

Name lengths
Request, macro, string, and number names can be longer than two letters, as in:

```
.html c <center>
.de footnote
Footnote here.
.. 
.footnote
.ds string "hello
*/[string]
.nr number 1
\n[number]
```

HTML output
Two new requests:

```
.html id [ <html>] 
.ihtml id [ <ihtml>
```

.html and .ihtml insert HTML into the output. The requests are only for opening new HTML tags. To close previously-opened tags, repeat the request with the same id. For example, the input:

```
.html t <table><tr>
.html td <td>Cell 1
.html td <td>Cell 2
.html t
```

produces this output:

```
<table><tr><td>Cell 1</td><td>Cell 2</td></tr></table>
```

The .html request is intended for block-level HTML constructs (those that can contain <p>) and maintains the HTML tag stack automatically. Intermediate tags need not be explicitly closed: removing the final .html t line in the example above would produce the same output. The special id – closes the HTML tags immediately after printing them.

The .ihtml request is similar to .html but is intended for inline HTML constructs such as <b> or <i> (those that can be contained within <p>). Unlike .html, .ihtml treats the open HTML tags as a set rather than a stack: each must be explicitly closed. Although it treats the tags as a set, .ihtml treats nesting properly in the output, closing and reopening tags as necessary. For example, the input:

```
.ihtml style <b>
.ihtml link <a href="link.html">Bold
.ihtml style <i>
and italic, still linked.
.ihtml link <a>
Unlinked.
.ihtml style
```

produces this output:

```
<b><a href="link.html">Bold</a></b>
<i><a href="link.html">and italic, still linked.</a></i>
<i>Unlinked.</i>
```
Outside of .html and .ihtml requests, the characters <, >, and & are treated as normal characters, not HTML markers, and are translated to &lt;, &gt;, and &amp; on output. To embed the raw HTML markers, use \<, \\\>, and @@ [sic].

Font changes
Htmlroff interprets the usual \f, \ft, \s, and \ps requests to change the font and point size. After applying each such change to its internal registers, htmlroff invokes the .font macro to emit corresponding HTML. The default definition of .font is:

```
.de font
.ihtml f1
.ihtml f
.ihtml f <span style=
.ihtml f1 <i>
.if \n(.f==2 .ihtml f1 <i>
.if \n(.f==3 .ihtml f1 <b>
.if \n(.f==4 .ihtml f1 <b>
.if \n(.f==5 .ihtml f1 <tt>
.if \n(.f==6 .ihtml f1 <tt><i>
```

Input files can redefine .font like any other request or macro.

Paragraphs
Htmlroff implements line height, text adjustment, and margins by wrapping all output text in <p style="..."> tags. This behavior can be disabled by setting the .paragraph number register to zero. Setting the .margin register to zero eliminates only the margin annotations.

Subscripts and superscripts
Htmlroff interprets the \u, \d, and \v requests to move vertically during output. It emits output vertically offset up the page inside <sup> tags and output vertically offset down the page inside <sub> tags. This heuristic handles simple equations formatted by eqn(1).

Conditional input
To make it easier to write input files that can be formatted by both troff and htmlroff, htmlroff adds a new condition \h which evaluates true in .if and .ie requests. The \t condition continues to evaluate true, to accommodate input files trying to distinguish between troff and nroff. To write a conditional matching troff alone, use `\if !h \if t`.

Htmlroff’s handling of conditional input does not match troff’s exactly. For example,

```
.if 0 \{
.de xx
```

redefines the xx macro in troff but not in htmlroff. Do not write files depending on this behavior, as this bug may be fixed in the future. Htmlroff also mishandles \} in some cases. To work around them, use \\} on a line by itself, as in the last example.

Diversions
Diversions in htmlroff use the alignment in effect at the time of the diversion when output. In particular,

```
.di xx
Line here.
..xx
```

produces a centered line in troff but not in htmlroff. The solution is to center inside the diversion, as in

```
.di xx
.if h .ce 999
Line here
..xx
```
Traps

*htmlroff* implements traps at vertical position 0, which run when the first character is about to be printed. Other position traps are ignored. Input traps are implemented.

Input pipes

*htmlroff* adds a new request `.inputpipe stop cmd` that redirects *htmlroff*'s input into a pipe to `cmd`. The redirection stops on encountering the line `stop`, optionally followed by white space and extra text. This is a dangerous and clumsy request, as *htmlroff* stops interpreting its input during the redirection, so `stop` must be found in the input itself, not in a macro that the input might appear to call. Although clumsy, `.inputpipe` allows input files to invoke *troff* to handle complicated input. For example, `tmac.html` redefines the PS macro that marks the beginning of a `pic(1)` picture:

```
.nr png −1 1
.de PS
.ds pngbase "\*[basename]
.if '\*[pngbase]' .ds pngbase \n(.B
.ds pngfile "\*[pngbase]\n+[png].png
.html − <center><img src="\*[pngfile]">\n</center>
.inputpipe .PE troff2png >\*[pngfile]
```

This macro invokes the shell script `troff2png` to run *troff* and convert the Postscript output to a PNG image file. Before starting the program, the macro creates a new file name for the image and prints HTML referring to it. The `.B` register holds the final path element (the base name) of the current input file.

Unimplemented

Tabs are set every eight spaces and cannot be changed.

Some requests, such as `.tl`, are unimplemented for lack of a good implementation. Workarounds can be defined as necessary in input files.

**SEE ALSO**

*htmlroff*(1), *mhtml*(6)
NAME
image - external format for images

SYNOPSIS
#include <draw.h>

DESCRIPTION
Images are described in graphics(2), and the definition of pixel values is in color(6). Fonts and
images are stored in external files in machine-independent formats.

Image files are read and written using readimage and writeimage (see allocimage(2), or
readmemimage and writememimage (see memdraw(2)). An uncompressed image file starts
with 5 strings: chan, r.min.x, r.min.y, r.max.x, and r.max.y. Each is right-justified
and blank padded in 11 characters, followed by a blank. The chan value is a textual string
describing the pixel format (see strtochan in graphics(2) and the discussion of channel
descriptors below), and the rectangle coordinates are decimal strings. The rest of the file contains
the r.max.y-r.min.y rows of pixel data. A row consists of the byte containing pixel
r.min.x and all the bytes up to and including the byte containing pixel r.max.x-1. For
images with depth d less than eight, a pixel with x-coordinate = x will appear as d contiguous bits
in a byte, with the pixel’s high order bit starting at the byte’s bit number w×(x mod (8/w)), where
bits within a byte are numbered 0 to 7 from the high order to the low order bit. Rows contain integral
number of bytes, so there may be some unused pixels at either end of a row. If d is greater
than 8, the definition of images requires that it will a multiple of 8, so pixel values take up an integral
number of bytes.

The loadimage and unloadimage functions described in allocimage(2) also deal with rows in
this format, stored in user memory.

The channel format string is a sequence of two-character channel descriptions, each comprising a
letter (r for red, g for green, b for blue, a for alpha, m for color-mapped, k for greyscale, and x
for “don’t care”) followed by a number of bits per pixel. The sum of the channel bits per pixel is
the depth of the image, which must be either a divisor or a multiple of eight. It is an error to have
more than one of any channel but x. An image must have either a greyscale channel; a color
mapped channel; or red, green, and blue channels. If the alpha channel is present, it must be at
least as deep as any other channel.

The channel string defines the format of the pixels in the file, and should not be confused with ordering of bytes in the file. In particular ’r8g8b8′ pixels have byte ordering blue, green, and red within the file. See color(6) for more details of the pixel format.

A venerable yet deprecated format replaces the channel string with a decimal ldepth, which is the base two logarithm of the number of bits per pixel in the image. In this case, ldepths 0, 1, 2, and 3 correspond to channel descriptors k1, k2, k4, and m8, respectively.

Compressed image files start with a line of text containing the word compressed, followed by a
header as described above, followed by the image data. The data, when uncompressed, is laid out
in the usual form.

The data is represented by a string of compression blocks, each encoding a number of rows of the
image’s pixel data. Compression blocks are at most 6024 bytes long, so that they fit comfortably
in a single 9P message. Since a compression block must encode a whole number of rows, there is
a limit (about 5825 bytes) to the width of images that may be encoded. Most wide images are in
subfonts, which, at 1 bit per pixel (the usual case for fonts), can be 46600 pixels wide.

A compression block begins with two decimal strings of twelve bytes each. The first number is
one more than the y coordinate of the last row in the block. The second is the number of bytes of
compressed data in the block, not including the two decimal strings. This number must not be
larger than 6000.

Pixels are encoded using a version of Lempel & Ziv’s sliding window scheme LZ77, best described
in J A Storer & T G Szymanski ‘Data Compression via Textual Substitution’, JACM 29#4, pp. 928–
951.

The compression block is a string of variable-length code words encoding substrings of the pixel
data. A code word either gives the substring directly or indicates that it is a copy of data occurring
previously in the pixel stream.

In a code word whose first byte has the high-order bit set, the rest of the byte indicates the length of a substring encoded directly. Values from 0 to 127 encode lengths from 1 to 128 bytes. Subsequent bytes are the literal pixel data.

If the high-order bit is zero, the next 5 bits encode the length of a substring copied from previous pixels. Values from 0 to 31 encode lengths from 3 to 34 bytes. The bottom two bits of the first byte and the 8 bits of the next byte encode an offset backward from the current position in the pixel data at which the copy is to be found. Values from 0 to 1023 encode offsets from 1 to 1024. The encoding may be 'prescient', with the length larger than the offset, which works just fine: the new data is identical to the data at the given offset, even though the two strings overlap.

Some small images, in particular 48×48 face files as used by `seemail` (see `faces(1)` and `face(6)`), and 16×16 cursors, can be stored textually, suitable for inclusion in C source. Each line of text represents one scan line as a comma-separated sequence of hexadecimal bytes, shorts, or words in C format. For cursors, each line defines a pair of bytes. (It takes two images to define a cursor; each must be stored separately to be processed by programs such as `tweak(1)`.) Face files of one bit per pixel are stored as a sequence of shorts, those of larger pixel sizes as a sequence of longs. Software that reads these files must deduce the image size from the input; there is no header. These formats reflect history rather than design.

SEE ALSO

`jpg(1)`, `tweak(1)`, `graphics(2)`, `draw(2)`, `allocimage(2)`, `color(6)`, `face(6)`, `font(6)`
NAME
keyboard – how to type characters

DESCRIPTION
Keyboards are idiosyncratic. It should be obvious how to type ordinary ASCII characters, backspace, tab, escape, and newline. In Plan 9, the key labeled Return or Enter generates a newline (0x0A); if there is a key labeled Line Feed, it generates a carriage return (0x0D); Plan 9 eschews CRLFs. All control characters are typed in the usual way; in particular, control-J is a line feed and control-M a carriage return. On the PC and some other machines, the key labeled Caps Lock acts as an additional control key.

The delete character (0x7F) may be generated by a different key, one near the extreme upper right of the keyboard. On the Next it is the key labeled * (not the asterisk above the 8). On the SLC and Sparcstation 2, delete is labeled Num Lock (the key above Backspace labeled Delete functions as an additional backspace key). On the other keyboards, the key labeled Del or Delete generates the delete character.

The view character (0x80), used by rio(1), acme(1), and sam(1), causes windows to scroll forward. It is generally somewhere near the lower right of the main key area. The scroll character is generated by the VIEW key on the Gnot, the Alt Graph key on the SLC, and the arrow key i on the other terminals. As a convenience for sloppy typists, some programs interpret −− and −− keys, which lie on either side of i, as view keys as well. The arrow key i scrolls backward.

Characters in Plan 9 are runes (see utf(6)). Any 16–bit rune can be typed using a compose key followed by several other keys. The compose key is also generally near the lower right of the main key area: the NUM PAD key on the Gnot, the Alternate key on the Next, the Compose key on the SLC, the Option key on the Magnum, and either Alt key on the PC. After typing the compose key, type a capital X and exactly four hexadecimal characters (digits and a to f) to type a single rune with the value represented by the typed number. There are shorthands for many characters, comprising the compose key followed by a two– or three–character sequence. There are several rules guiding the design of the sequences, as illustrated by the following examples. The full list is too long to repeat here, but is contained in the file /lib/keyboard in a format suitable for grep(1) or look(1).

A repeated symbol gives a variant of that symbol, e.g., ?? yields ±.

ASCII digraphs for mathematical operators give the corresponding operator, e.g., <= yields ≤.

Two letters give the corresponding ligature, e.g., AE yields A.

Mathematical and other symbols are given by abbreviations for their names, e.g., pg yields ¶.

Chess pieces are given by a w or b followed by a letter for the piece (k for king, q for queen, r for rook, n for knight, b for bishop, or p for pawn), e.g., wk for a white king.

Greek letters are given by an asterisk followed by a corresponding latin letter, e.g., *d yields δ.

Cyrillic letters are given by an at sign followed by a corresponding latin letter or letters, e.g., @ya yields ¥.

Script letters are given by a dollar sign followed by the corresponding regular letter, e.g., $F yields F.

A digraph of a symbol followed by a letter gives the letter with an accent that looks like the symbol, e.g., â yields ȧ.

Two digits give the fraction with that numerator and denominator, e.g., 12 yields ½.

The letter s followed by a character gives that character as a superscript, e.g., s1 yields 1.

These characters are taken from the Unicode block 0x2070; the 1, 2, and 3 superscripts in the Latin–1 block are available by using a capital S instead of s.

Sometimes a pair of characters give a symbol related to the superimposition of the characters, e.g., c0 yields ©.
A mnemonic letter followed by $ gives a currency symbol, e.g., £$ yields £.
Note the difference between ß (ss) and µ (micron) and the Greek β and µ.

FILES
/lib/keyboard     sorted table of characters and keyboard sequences

SEE ALSO
intro(1), ascii(1), tcs(1), acme(1), rio(1), sam(1), cons(3), utf(6)
NAME
    keys.who - biographic information for key holders

DESCRIPTION
    When auth/changeuser (see auth(8)) creates or modifies an account, it writes a line of biographical
    information to /adm/keys.who. The line contains the following fields, separated by | characters:
       name    login name
       postid  company-wide user name
       full name    full name of the user
depart    department of the user
       email...    one or more fields containing email addresses to be notified when the key is about to
                    expire

    The program auth/warning, which has fallen into disrepair, once read keys.who and mailed expiry
    warnings.

EXAMPLE
    rsc|rsc|ox|Russell S Cox|11276|rsc|dmr|rob

SEE ALSO
    keyfs(4), auth(8)
NAME
man - macros to typeset manual

SYNOPSIS
nroff  -man file ... 
troff  -man file ...

DESCRIPTION
These macros are used to format pages of this manual.
Except in .LR and .RL requests, any text argument denoted \t in the request summary may be
zero to six words. Quotes " ... " may be used to include blanks in a 'word'. If \t is empty, the
special treatment is applied to the next text input line (the next line that doesn’t begin with dot).
In this way, for example, .I may be used to italicize a line of more than 6 words, or .SM followed
by .B to make small letters in 'bold' font.
A prevailing indent distance is remembered between successive indented paragraphs, and is reset
to default value upon reaching a non-indented paragraph. Default units for indents \i are ens.
The fonts are
\R roman, the main font, preferred for diagnostics
\I italic, preferred for parameters, short names of commands, names of manual pages, and
naked function names
\B 'bold', actually the constant width font, preferred for examples, file names, declarations,
keywords, names of struct members, and literals (numbers are rarely literals)
\L also the constant width font. In troff L=B; in nroff arguments of the macros .L, .LR, and
\RL are printed in quotes; preferred only where quotes really help (e.g. lower-case literals
and punctuation).

Type font and size are reset to default values before each paragraph, and after processing font-or
size-setting macros.
The -man macros admit equations and tables in the style of eqn(1) and tbl(1), but do not support
arguments on .EQ and .TS macros.
These strings are predefined by -man:
\*R 'e', '(Reg)' in nroff.
\*S Change to default type size.

FILES
/sys/lib/tmac/tmac.an

SEE ALSO
troff(1), man(1)

REQUESTS
Request Cause If no Break Argument Explanation
.B t no \t=n.t.l.* Text \t is 'bold'.
.BI t no \t=n.t.l. Join words of \t alternating bold and italic.
.BR t no \t=n.t.l. Join words of \t alternating bold and Roman.
.DT no Restore default tabs.
.EE yes End displayed example
.EX yes Begin displayed example
.HP i yes \i=p.i.* Set prevailing indent to \i. Begin paragraph with hanging indent.
.I t no \t=n.t.l. Text \t is italic.
.IB t no \t=n.t.l. Join words of \t alternating italic and bold.
.IP x i yes \x=\" Same as .IP with tag \x.
.IR t no \t=n.t.l. Join words of \t alternating italic and Roman.
.L t no \t=n.t.l. Text \t is literal.
.LP yes Same as .PP.
.LR t no Join 2 words of \t alternating literal and Roman.
.PD d no \d=.4v Interparagraph distance is \d.
.PP yes Begin paragraph. Set prevailing indent to default.
.RE yes End of relative indent. Set prevailing indent to amount of starting .RS.
.RI t no t=n.t.l. Join words of t alternating Roman and italic.
.RL t no Join 2 or 3 words of t alternating Roman and literal.
.RS i yes i=p.i. Start relative indent, move left margin in distance i. Set prevailing indent to
default for nested indents.
.SH t yes t= Subhead; reset paragraph distance.
.SM t no t=n.t.l. Text t is small.
_SS t no t= Secondary subhead.
.TF s yes Prevailing indent is wide as string s in font L; paragraph distance is 0.
.TH n c x yes Begin page named n of chapter c; x is extra commentary, e.g. ‘local’, for page
head. Set prevailing indent and tabs to default.
.TP i yes i=p.i. Set prevailing indent to i. Restore default indent if i=0. Begin indented para-
graph with hanging tag given by next text line. If tag doesn’t fit, place it on
separate line.
.1C yes Equalize columns and return to 1–column output
.2C yes Start 2–column nofill output

* n.t.l. = next text line; p.i. = prevailing indent

**BUGS**

There’s no way to fool troff into handling literal double quote marks ” in font-alternation macros,
such as .BI.
There is no direct way to suppress column widows in 2–column output; the column lengths may be
adjusted by inserting . sp requests before the closing .1C.
NAME
map - digitized map formats

DESCRIPTION
Files used by map(7) are a sequence of structures of the form:

```
struct {
    signed char patchlatitude;
    signed char patchlongitude;
    short n;
    union {
        struct {
            short latitude;
            short longitude;
        } point[n];
        struct {
            short latitude;
            short longitude;
            struct {
                signed char latdiff;
                signed char londiff;
            } point[-n];
        } highres;
    } segment;
};
```

where short stands for 16-bit integers and there is no padding within or between structs. Shorts are stored in little-endian order, low byte first. To assure portability, map accesses them bytewise.

Fields patchlatitude and patchlongitude tell to what 10-degree by 10-degree patch of the earth's surface a segment belongs. Their values range from -9 to 8 and from -18 to 17, respectively, and indicate the coordinates of the southeast corner of the patch in units of 10 degrees.

Each segment of |n| points is connected; consecutive segments are not necessarily related. Latitude and longitude are measured in units of 0.0001 radian. If n is negative, then differences to the first and succeeding points are measured in units of 0.00001 radian. Latitude is counted positive to the north and longitude positive to the west.

The patches are ordered lexicographically by patchlatitude then patchlongitude. A printable index to the first segment of each patch in a file named data is kept in an associated file named data.x. Each line of an index file contains patchlatitude, patchlongitude and the byte position of the patch in the map file. Both the map file and the index file are ordered by patch latitude and longitude.

SEE ALSO
map(7)
The data comes from the World Data Bank I and II and U.S. Government sources: the Census Bureau, Geological Survey, and CIA.
NAME
    mhtml - macros for formatting HTML

SYNOPSIS
    pic | tbl | eqn | htmlroff [-man | -ms] -mhtml file ...

DESCRIPTION
    This package of htmlroff(1) macro definitions provides convenient macros for formatting HTML. It is usually used along with troff(1) macro packages such as man(6) and ms(6). Mhtml replaces some macros defined in the other packages, so it should be listed after them on the htmlroff command line.

    The following macros are defined:

    .HTML title
        Print an HTML header marking the output as HTML 4.01 loose transitional encoded in UTF. If given, the title is printed inside <title> tags. This macro opens the <html> tag, opens and closes the <head> section, and opens <body>. It invokes the .HEAD macro inside the <head> section. To add arbitrary lines to the header, append to .HEAD before invoking .HTML.

    .FS, .FE
        Accumulate footnotes and print them at the end of the document under a Notes heading. These replace the macros in ms(6). To emit the notes accumulated so far, invoke .NOTES.

    .PS, .PE
        Replace input bracketed .PS and .PE with a PNG image corresponding to the output of running troff(1) on the input.

    .TS, .TE
        Identical to .PS and .PE.

    .B1 margin width, .B2
        Format the input between .B1 and .B2 inside a box, with margin (default 10) pixels between the box and the text. The box is set to be width (default 60) percent of the current output width.

FILES
    /sys/lib/tmac/tmac.html

SEE ALSO
    htmlroff(1), htmlroff(6), ms(6)
NAME
mnihongo - macros for typesetting Japanese

SYNOPSIS
troff -mnihongo ...

DESCRIPTION
Mnihongo provides a simple troff(1) post-processor that formats Unicode characters that might be Japanese text. It looks up the characters in the bitmap font /lib/font/bit/pelm/unicode.9x24.font and generates bitmap images embedded in the output.

During troff processing, widths of the Japanese characters are taken from the troff font Jp, which is at best a simple approximation to the truth.

FILES
/bin/aux/mnihongo
/sys/lib/tmac/tmac.nihongo
/lib/font/bit/pelm/unicode.9x24.font

SOURCE
/sys/src/cmd/aux/mnihongo

SEE ALSO
troff(1)
NAME
mpictures - picture inclusion macros

SYNOPSIS
troff -mpictures [ options ] file ...

DESCRIPTION
Mpictures macros insert PostScript pictures into troff(1) documents. The macros are:

.BP source height width position offset flags label
Define a frame and place a picture in it. Null arguments, represented by " ", are interpreted as defaults. The arguments are:
source Name of a PostScript picture file, optionally suffixed with (n) to select page number n from the file (first page by default).
height Vertical size of the frame, default 3.0i.
width Horizontal size of the frame, current line length by default.
position
1 (default), c, or r to left-justify, center, or right-justify the frame.
offset Move the frame horizontally from the original position by this amount, default 0i.
flags One or more of:
ad Rotate the picture clockwise d degrees, default d=90.
o Outline the picture with a box.
s Freely scale both picture dimensions.
w White out the area to be occupied by the picture.
l,r,t,b
Attach the picture to the left right, top, or bottom of the frame.
label Place label at distance 1.5v below the frame.

If there's room, .BP fills text around the frame. Everything destined for either side of the frame goes into a diversion to be retrieved when the accumulated text sweeps past the trap set by .BP or when the diversion is explicitly closed by .EP.

.PI source height, width, yoffset, xoffset flags.
This low-level macro, used by .BP, can help do more complex things. The two arguments not already described are:
xoffset Offset the frame from the left margin by this amount, default 0i.
yoffset Offset the frame from the current baseline, measuring positive downward, default 0i.

.EP End a picture started by .BP; .EP is usually called implicitly by a trap at frame bottom.

If a PostScript file lacks page-delimiting comments, the entire file is included. If no %BoundingBox comment is present, the picture is assumed to fill an 8.5×11-inch page. Nothing prevents the picture from being placed off the page.

SEE ALSO
troff(1)

DIAGNOSTICS
A picture file that can't be read by the PostScript postprocessor is replaced by white space.

BUGS
A picture and associated text silently disappear if a diversion trap set by .BP isn't reached. Call .EP at the end of the document to retrieve it.
Macros in other packages may break the adjustments made to the line length and indent when text is being placed around a picture.
A missing or improper %BoundingBox comment may cause the frame to be filled incorrectly.
NAME
ms - macros for formatting manuscripts

SYNOPSIS
  nroff  -ms [ options ] file ...
  troff  -ms [ options ] file ...

DESCRIPTION
This package of nroff and troff(1) macro definitions provides a canned formatting facility for technical papers in various formats.

The macro requests are defined below. Many nroff and troff requests are unsafe in conjunction with this package, but the following requests may be used with impunity after the first .PP: .bp, .br, .sp, .ls, .na.

Output of the eqn(1), tbl(1), pic(1) and grap(1) preprocessors for equations, tables, pictures, and graphs is acceptable as input.

FILES
/sys/lib/tmac/tmac.s

SEE ALSO
eqn(1), troff(1), tbl(1), pic(1)

REQUESTS
Request Initial Cause Explanation
  Value Break
.
  .1C yes yes One column format on a new page.
.
  .2C no yes Two column format.
.
  .AB no yes Begin abstract.
.
  .AE - yes End abstract.
.
  .AI no yes Author’s institution follows. Suppressed in .TM.
.
  .AT no yes Print ‘Attached’ and turn off line filling.
.
  .AU x y no yes Author’s name follows. x is location and y is extension, ignored except in .TM.
.
  .B x y z no no Print x in boldface, append roman y and preface with z; if no argument switch to boldface.
.
  .B1 no yes Begin text to be enclosed in a box.
.
  .B2 no yes End boxed text.
.
  .B1 x y z no no Print x in bold italic, append roman y and preface with z; if no argument switch to bold italic.
.
  .BT date no Bottom title, automatically invoked at foot of page. May be redefined.
.
  .BX x no no Print x in a box.
.
  .CW x y z no no Constant width font for x, append roman y and preface with z; if no argument switch to constant width.
.
  .CT no yes Print ‘Copies to’ and turn off line filling.
.
  .DA x nroff no ‘Date line’ at bottom of page is x. Default is today.
.
  .DE - yes End displayed text. Implies .KE.
.
  .DS x no yes Start of displayed text, to appear verbatim line-by-line: I indented (default), L left-justified, C centered, B (block) centered with straight left margin. Implies .KS.
.
  .EG no - Print document in BTL format for ‘Engineer’s Notes.’ Must be first.
.
  .EN - yes Space after equation produced by neqn or eqn(1).
.
  .EQ x y - yes Display equation. Equation number is y. Optional x is I, L, C as in .DS.
.
  .FE - yes End footnote.
.
  .FP x - no Set font positions for a family, e.g., .FP lucidasans
.
  .FS no no Start footnote. The note will be moved to the bottom of the page.
.
  .HO - no ‘Bell Laboratories, Holmdel, New Jersey 07733’.
.
  .I x y z no no Italicize x, append roman y and preface with z; if no argument switch to italic.
.
  .IH no no ‘Bell Laboratories, Naperville, Illinois 60540’
.
  .IM no no Print document in BTL format for an internal memorandum. Must be first.
.IP x y no yes Start indented paragraph, with hanging tag x. Indentation is y ens (default 5).
.KE – yes End keep. Put kept text on next page if not enough room.
.KF no yes Start floating keep. If the kept text must be moved to the next page, float later text back to this page.
.KS no yes Start keeping following text.
.LG no no Make letters larger.
.LP yes yes Start left–blocked paragraph.
.LT no yes Start a letter; a non–empty first argument produces a full Lucent letterhead, a second argument is a room number, a third argument is a telephone number.
.MF – – Print document in BTL format for ‘Memorandum for File.’ Must be first.
.MH no ‘Bell Laboratories, Murray Hill, New Jersey 07974’
.MR – – Print document in BTL format for ‘Memorandum for Record.’ Must be first.
.ND date troff no Use date supplied (if any) only in special BTL format positions; omit from page footer.
.NH n – yes Same as .SH, with automatic section numbers like ’1.2.3’; n is subsection level (default 1). If n is 0, reset the numbering.
.NL yes no Make letters normal size.
.P1 – yes Begin program display in constant width font.
.P2 – yes End program display.
.PE – yes End picture; see pic(1).
.PF – yes End picture; restore vertical position.
.PP no yes Begin paragraph. First line indented.
.PS h w – yes Start picture; height and width in inches.
.PY – no ‘Bell Laboratories, Piscataway, New Jersey 08854’
.QE – yes End quoted material.
.QP – yes Begin quoted paragraph (indent both margins).
.QS – yes Begin quoted material (indent both margins).
.R yes no Roman text follows.
.RE – yes End relative indent level.
.RP no – Cover sheet and first page for released paper. Must precede other requests.
.RS – yes Start level of relative indentation from which subsequent indentation is measured.
.SG x no yes Insert signature(s) of author(s), ignored except in .TM and .LT. x is the reference line (initials of author and typist). .JF
.SH – yes Section head follows, font automatically bold.
.SM no no Make letters smaller.
.TA x... 5... no Set tabs in ens. Default is 5 10 15 ...
.TE – yes End table; see tbl(1).
.TH – yes End heading section of table.
.TL no yes Title follows.
.TM x... no – Print document in BTL technical memorandum format. Arguments are TM number, (quoted list of) case number(s), and file number. Must precede other requests.
.TR x – – Print in BTL technical report format; report number is x. Must be first.
.TS x yes Begin table; if x is H table heading is repeated on new pages.
.UL x – no Underline argument (even in troff).
.UX y z – no ‘2UNIX’; first use gives registered trademark notice.
.WH – no ‘Bell Laboratories, Whippany, New Jersey 07981’.
NAME
  namespace – name space description file

DESCRIPTION
Namespace files describe how to construct a name space from scratch, an operation normally performed by the newns or addns subroutines (see auth(2)) which is typically called by init(8). Each line specifies one name space operation. Spaces and tabs separate arguments to operations; no quotes or escapes are recognized. Blank lines and lines with # as the first non-space character are ignored. Environment variables of the form $name are expanded within arguments, where name is a UTF string terminated by white space, a /, or a $.

The known operations and their arguments are:

  mount [-abcC] servename old [spec]
  Mount servename on old.

  bind [-abcC] new old
  Bind new on old.

  import [-abc] host [remotepath] mountpoint
  Import remotepath from machine server and attach it to mountpoint.

  cd dir
  Change the working directory to dir.

  unmount [new] old
  Unmount new from old, or everything mounted on old if new is missing.

  clear
  Clear the name space with rfork(RFCNAMEG).

  . path
  Execute the namespace file path. Note that path must be present in the name space being built.

The options for bind, mount, and import are interpreted as in bind(1) and import(4).

SEE ALSO
  bind(1), namespace(4), init(8)
NAME
ndb - Network database

DESCRIPTION
The network database consists of files describing machines known to the local installation and machines known publicly. The files comprise multi-line tuples made up of attribute/value pairs of the form attr=value or sometimes just attr. Each line starting without white space starts a new tuple. Lines starting with # are comments.

The file /lib/ndb/local is the root of the database. Other files are included in the database if a tuple with an attribute-value pair of attribute database and no value exists in /lib/ndb/local. Within the database tuple, each pair with attribute file identifies a file to be included in the database. The files are searched in the order they appear. For example:

database=
  file=/lib/ndb/common
  file=/lib/ndb/local
  file=/lib/ndb/global

declares the database to be composed of the three files /lib/ndb/common, /lib/ndb/local, and /lib/ndb/global. By default, /lib/ndb/local is searched before the others. However, /lib/ndb/local may be included in the database to redefine its ordering.

Within tuples, pairs on the same line bind tighter than pairs on different lines.

Programs search the database directly using the routines in ndb(2) or indirectly using ndb/cs and ndb/dns (see ndb(8)). Both ndb/cs and the routine ndbinfo impose structure on the otherwise flat database by using knowledge specific to the network. The internet is made up of networks which can be subnetted multiple times. A network must have an ipnet attribute and is uniquely identified by the values of its ip and ipmask attributes. If the ipmask is missing, the relevant Class A, B or C one is used.

A search for an attribute associated with a network or host starts at the lowest level, the entry for the host or network itself, and works its way up, bit by bit, looking at entries for nets/subnets that include the network or host. The search ends when the attribute is found. For example, consider the following entries:

ipnet=murray−hill ip=135.104.0.0 ipmask=255.255.0.0
  dns=135.104.10.1
  ntp=ntp.cs.bell−labs.com
ipnet=plan9 ip=135.104.9.0 ipmask=255.255.255.0
  ntp=oncore.cs.bell−labs.com
  smtp=smtp1.cs.bell−labs.com
ip=135.104.9.6 sys=anna dom=anna.cs.bell−labs.com
  smtp=smtp2.cs.bell−labs.com

Here anna is on the subnet plan9 which is in turn on the class B net murray−hill. Assume that we're searching for anna's NTP and SMTP servers. The search starts by looking for an entry with sys=anna. We find the anna entry. Since it has an smtp=smtp2.cs.bell−labs.com pair, we're done looking for that attribute. To fulfill the NTP request, we continue by looking for networks that include anna's IP address. We lop off the right most one bit from anna's address and look for an ipnet= entry with ip=135.104.9.4. Not finding one, we drop another bit and look for an ipnet= entry with ip=135.104.9.0. There is such an entry and it has the pair, ntp=oncore.cs.bell−labs.com, ending our search.

Ndb/cs can be made to perform such network aware searches by using metanames in the dial-string. A metaname is a $ followed by an attribute name. Ndb/cs looks up the attribute relative to the system it is running on. Thus, with the above example, if a program called

dial("tcp!$smtp!smtp", 0, 0, 0);

the dial would connect to the SMTP port of smtp2.cs.bell−labs.com.

A number of attributes are meaningful to programs and thus reserved. They are:
sys    system name (a short name)
dom   Internet fully-qualified domain name
ip    Internet address, v4 or v6.
ipv6  IPv6 Internet address. For DNS, an AAAA record.
ether Ethernet address (must be lower-case hexadecimal). Beware that for machines with multiple ether attributes, dhcpd may expect requests to come from the address in the first ether attribute.
bootf file to download for initial bootstrap: /386/9pxeboot to boot a PC via PXE.
ipnet Internet network name
ipmask Internet network mask
ipgw  Internet gateway
auth  authentication server to be used
authdom authentication domain. Plan 9 supports multiple authentication domains. To specify an authentication server for a particular domain, add a tuple containing both auth and authdom attributes and values.
fs    file server to be used
tcp   a TCP service name
udp   a UDP service name
port  a TCP or UDP port number
restricted a TCP service that can be called only by ports numbered less that 1024
proto a protocol supported by a host. The pair proto=il was needed by cs (see ndb(8)) in tuples for hosts that supported the IL protocol
dnsdomain a domain name that ndb/dns adds onto any unrooted names when doing a search. There may be multiple dnsdomain pairs.
dns   a DNS server to use (for DNS and DHCP)
tcp   an NTP server to use (for DHCP)
smtp  an SMTP server to use (for DHCP)
time a time server to use (for DHCP)
wins  a Windows name server (for DHCP)
mx    mail exchanger (for DNS and DHCP); also pref.
srv   service location (for DNS); also pri, weight and port.
soa   start of area (for DNS)

Cs defers to dns to translate dotted names to IP addresses, only consulting the database files if dns cannot translate the name.

Cs allows network entries with sys and dom attributes but no ip attribute. Searches for the system name are resolved by looking up the domain name with dns.

The file /lib/ndb/auth is used during authentication to decide who has the power to ‘speak for’ other users; see authsrv(6).

EXAMPLES
A tuple for the CPU server, spindle.

sys=spindle
dom=spindle.research.bell-labs.com
bootf=/mips/9powerboot
ip=135.104.117.32 ether=08006920677

ipnet=mh–astro–net ip=135.104.0.0 ipmask=255.255.255.0
fs=bootes.research.bell-labs.com
ipgw=r70.research.bell-labs.com
auth=p9auth.research.bell-labs.com
ipnet=unix–room ip=135.104.117.0
ipgw=135.104.117.1
ipnet=third–floor ip=135.104.51.0
ipgw=135.104.51.1

Mappings between TCP service names and port numbers.
tcp=sysmon port=401
tcp=rexec port=512 restricted
tcp=9fs    port=564

FILES
    /lib/ndb/local first database file searched

SEE ALSO
    con(1), dial(2), ndb(2), 9load(8), booting(8), dhcpd(8), ipconfig(8), ndb(8)
NAME
plot - graphics interface

DESCRIPTION
Files of this format are interpreted by plot(1) to draw graphics on the screen. A plot file is a UTF stream of instruction lines. Arguments are delimited by spaces, tabs, or commas. Numbers may be floating point. Punctuation marks (except :) , spaces, and tabs at the beginning of lines are ignored. Comments run from : to newline. Extra letters appended to a valid instruction are ignored. Thus ... line, line, line all mean the same thing. Arguments are interpreted as follows:

1. If an instruction requires no arguments, the rest of the line is ignored.
2. If it requires a string argument, then all the line after the first field separator is passed as argument. Quote marks may be used to preserve leading blanks. Strings may include newlines represented as \
.
3. Between numeric arguments alphabetic characters and punctuation marks are ignored.

Thus line from 5 6 to 7 8 draws a line from (5, 6) to (7, 8).
4. Instructions with numeric arguments remain in effect until a new instruction is read. Such commands may spill over many lines. Thus the following sequence will draw a polygon with vertices (4.5, 6.77), (5.8, 5.6), (7.8, 4.55), and (10.0, 3.6).

move 4.5 6.77
vec 5.8, 5.6 7.8
4.55 10.0, 3.6 4.5, 6.77

The instructions are executed in order. The last designated point in a line, move, rmove, vec, rvec, arc, or point command becomes the ‘current point’ (X,Y) for the next command.

Open & Close
o string Open plotting device. For troff, string specifies the size of the plot (default is 6i).
c1 Close plotting device.

Basic Plotting Commands
e Start another frame of output.
m x y (move) Current point becomes x y.
rm dx dy Current point becomes X+dx Y+dy.
pol x y Plot the point x y and make it the current point.
v x y Draw a vector from the current point to x y.
rv dx dy Draw vector from current point to X+dx Y+dy
li x1 y1 x2 y2
Draw a line from x1 y1 to x2 y2.
	 string Place the string so that its first character is centered on the current point (default). If string begins with \ (\R), it is centered (right-adjusted) on the current point. A backslash at the beginning of the string may be escaped with another backslash.

a x1 y1 x2 y2 xc yc r
Draw a circular arc from x1 y1 to x2 y2 with center xc yc and radius r. If the radius is positive, the arc is drawn counterclockwise; negative, clockwise. The starting point is exact but the ending point is approximate.

ci xc yc r
Draw a circle centered at xc yc with radius r. If the range and frame parameters do not specify a square, the ‘circle’ will be elliptical.

di xc yc r
Draw a disc centered at xc yc with radius r using the filling color (see cfill1 below).

bo x1 y1 x2 y2
Draw a box with lower left corner at x1 y1 and upper right corner at x2 y2.

sb x1 y1 x2 y2
Draw a solid box with lower left corner at x1 y1 and upper right corner at x2 y2 using the filling color (see cfill1 below).

par x1 y1 x2 y2 xg yg
Draw a parabola from x1 y1 to x2 y2 ‘guided’ by xg yg. The parabola passes through the
midpoint of the line joining $xg\ yg$ with the midpoint of the line joining $x1\ y1$ and $x2\ y2$
and is tangent to the lines from $xg\ yg$ to the endpoints.

$$\text{pol} \{ \{ x1\ y1 ... \ xn\ yn\} ... \{ X1\ Y1 ... Xm\ Ym\} \}$$
Draw polygons with vertices $x1\ y1 ... \ xn\ yn$ and $X1\ Y1 ... Xm\ Ym$. If only one polygon is
specified, the inner brackets are not needed.

$$\text{fi} \{ \{ x1\ y1 ... \ xn\ yn\} ... \{ X1\ Y1 ... Xm\ Ym\} \}$$
Fill a polygon. The arguments are the same as those for pol except that the first vertex
is automatically repeated to close each polygon. The polygons do not have to be con-
nected. Enclosed polygons appear as holes.

$$\text{sp} \{ \{ x1\ y1 ... \ xn\ yn\} ... \{ X1\ Y1 ... Xm\ Ym\} \}$$
Draw a parabolic spline guided by $x1\ y1 ... \ xn\ yn$ with simple endpoints.

$$\text{fsp} \{ \{ x1\ y1 ... \ xn\ yn\} ... \{ X1\ Y1 ... Xm\ Ym\} \}$$
Draw a parabolic spline guided by $x1\ y1 ... \ xn\ yn$ with double first endpoint.

$$\text{lsp} \{ \{ x1\ y1 ... \ xn\ yn\} ... \{ X1\ Y1 ... Xm\ Ym\} \}$$
Draw a parabolic spline guided by $x1\ y1 ... \ xn\ yn$ with double last endpoint.

$$\text{dsp} \{ \{ x1\ y1 ... \ xn\ yn\} ... \{ X1\ Y1 ... Xm\ Ym\} \}$$
Draw a parabolic spline guided by $x1\ y1 ... \ xn\ yn$ with double endpoints.

in $\text{filename}$
(include) Take commands from $\text{filename}$.

def $\text{string} \{ \text{commands} \}$$
Define $\text{string}$ as $\text{commands}$.

call $\text{string} \{ \text{commands} \}$$
Invoke commands defined as $\text{string}$ applying $\text{scale}$ to all coordinates.

**Commands Controlling the Environment**

co $\text{string}$
Use color given by first character of $\text{string}$, one of red, yellow, green, blue,
cyan, magenta, white, and kblack.

pe $\text{string}$
Use $\text{string}$ as the style for drawing lines. The available pen styles are: solid,
dotted, short, long, dotted[ashed], cdash, ddash

cf $\text{string}$
Color for filling (see co, above).

ra $x1\ y1 x2\ y2$
The data will fall between $x1\ y1$ and $x2\ y2$. The plot will be magnified or reduced to fit
the device as closely as possible.

Range settings that exactly fill the plotting area with unity scaling appear below for
devices supported by the filters of plot(). The upper limit is just outside the plotting
area. In every case the plotting area is taken to be square; points outside may be dis-
playable on devices with nonsquare faces.

fr $px1\ py1 px2\ py2$
Plot the data in the fraction of the display specified by $px1\ py1$ for lower left corner and
px2\ py2 for upper right corner. Thus frame .5 0 1. .5 plots in the lower right
quadrant of the display; frame 0 .1 .1 0. uses the whole display but inverts the
y coordinates.

sa
Save the current environment, and move to a new one. The new environment inherits
the old one. There are 7 levels.

re
Restore previous environment.

**SEE ALSO**
plot(1), graph(1)
NAME
plumb - format of plumb messages and rules

SYNOPSIS
#include <plumb.h>

DESCRIPTION
Message format
The messages formed by the plumb(2) library are formatted for transmission between processes into textual form, using newlines to separate the fields. Only the data field may contain embedded newlines. The fields occur in a specified order, and each has a name, corresponding to the elements of the Plumbmsg structure, that is used in the plumbing rules. The fields, in order, are:

src application/service generating message
dst destination ‘port’ for message
wdir working directory (used if data is a file name)
type form of the data, e.g. text
attr attributes of the message, in name=value pairs separated by white space (the value must follow the usual quoting convention if it contains white space or quote characters or equal signs; it cannot contain a newline)
ndata number of bytes of data
data the data itself

At the moment, only textual data (type=text) is supported.

All fields are optional, but type should usually be set since it describes the form of the data, and ndata must be an accurate count (possibly zero) of the number of bytes of data. A missing field is represented by an empty line.

Plumbing rules
The plumber (see plumb(2)) receives messages on its send port (applications send messages there), interprets and reformats them, and (typically) emits them from a destination port. Its behavior is determined by a plumbing rules file, default /usr/$USER/lib/plumbing, which defines a set of pattern/action rules with which to analyze, rewrite, and dispatch received messages.

The file is a sequence of rule sets, each of which is a set of one–line rules called patterns and actions. There must be at least one pattern and one action in each rule set. (The only exception is that a rule set may contain nothing but plumb to rules; such a rule set declares the named ports but has no other effect.) A blank line terminates a rule set. Lines beginning with a # character are commentary and are regarded as blank lines.

A line of the form
include file
substitutes the contents of file for the line, much as in a C #include statement. Unlike in C, the file name is not quoted. If file is not an absolute path name, or one beginning ./ or ../, file is looked for first in the directory in which the plumber is executing, and then in /sys/lib/plumb.

When a message is received by the plumber, the rule sets are examined in order. For each rule set, if the message matches all the patterns in the rule set, the actions associated with the rule set are triggered to dispose of the message. If a rule set is triggered, the rest are ignored for this message. If none is triggered, the message is discarded (giving a write error to the sender) unless it has a dst field that specifies an existing port, in which case the message is emitted, unchanged, from there.

Patterns and actions all consist of three components: an object, a verb, and arguments. These are separated by white space on the line. The arguments may contain quoted strings and variable substitutions, described below, and in some cases contain multiple words. The object and verb are single words from a pre–defined set.

The object in a pattern is the name of an element of the message, such as src or data, or the special case arg, which refers to the argument component of the current rule. The object in an action is always the word plumb.
The verbs in the pattern rules describe how the objects and arguments are to be interpreted. Within a rule set, the patterns are evaluated in sequence; if one fails, the rule set fails. Some verbs are predicates that check properties of the message; others rewrite components of the message and implicitly always succeed. Such rewritings are permanent, so rules that specify them should be placed after all pattern-matching rules in the rule set.

add The object must be attr. Append the argument, which must be a sequence of name=value pairs, to the list of attributes of the message.

delete The object must be attr. If the message has an attribute whose name is the argument, delete it from the list of attributes of the message. (Even if the message does not, the rule matches the message.)
is If the text of the object is identical to the text of the argument, the rule matches.
isdir If the text of the object is the name of an existing directory, the rule matches and sets the variable $dir to that directory name.
isfile If the text of the object is the name of an existing file (not a directory), the rule matches and sets the variable $file to that file name.
matches If the entire text of the object matches the regular expression specified in the argument, the rule matches. This verb is described in more detail below.

set The value of the object is set to the value of the argument.

The matches verb has special properties that enable the rules to select which portion of the data is to be sent to the destination. By default, a data matches rule requires that the entire text matches the regular expression. If, however, the message has an attribute named click, that reports that the message was produced by a mouse click within the text and that the regular expressions in the rule set should be used to identify what portion of the data the user intended. Typically, a program such as an editor will send a white-space delimited block of text containing the mouse click, using the value of the click attribute (a number starting from 0) to indicate where in the textual data the user pointed.

When the message has a click attribute, the data matches rules extract the longest leftmost match to the regular expression that contains or abuts the textual location identified by the click. For a sequence of such rules within a given rule set, each regular expression, evaluated by this specification, must match the same subset of the data for the rule set to match the message. For example, here is a pair of patterns that identify a message whose data contains the name of an existing file with a conventional ending for an encoded picture file:

data matches '^[a-zA-Z0-9_-.]+$'
data matches '^[a-zA-Z0-9_-.]+\.(jpe?g|gif|bit|ps|pdf)$'

The first expression extracts the largest subset of the data around the click that contains file name characters; the second sees if it ends with, for example, .jpeg. If only the second pattern were present, a piece of text horse.gif could be misinterpreted as an image file named horse.gif.

If a click attribute is specified in a message, it will be deleted by the plumber before sending the message if the data matches rules expand the selection.

The action rules all have the object plumber. There are only three verbs for action rules:

to The argument is the name of the port to which the message will be sent. If the message has a destination specified, it must match the to port of the rule set or the entire rule set will be skipped. (This is the only rule that is evaluated out of order.)

client If no application has the port open, the arguments to a plumber start rule specify a shell program to run in response to the message. The message will be held, with the supposition that the program will eventually open the port to retrieve it.

start Like client, but the message is discarded. Only one start or client rule should be specified in a rule set.

The arguments to all rules may contain quoted strings, exactly as in rc(1). They may also contain simple string variables, identified by a leading dollar sign $. Variables may be set, between rule sets, by assignment statements in the style of rc. Only one variable assignment may appear on a line. The plumber also maintains some built-in variables:

$0 The text that matched the entire regular expression in a previous data matches rule. $1, $2, etc. refer to text matching the first, second, etc. parenthesized subexpression.
$attr  The textual representation of the attributes of the message.
$data  The contents of the data field of the message.
$dir   The directory name resulting from a successful isdir rule. If no such rule has
       been applied, it is the string constructed syntactically by interpreting data as a
       file name in wdir.
$dst   The contents of the dst field of the message.
$file  The file name resulting from a successful isfile rule. If no such rule has been
       applied, it is the string constructed syntactically by interpreting data as a file
       name in wdir.
$type  The contents of the type field of the message.
$src   The contents of the src field of the message.
$wdir  The contents of the wdir field of the message.

EXAMPLE
The following is a modest, representative file of plumbing rules.
# these are generally in order from most specific to least,
# since first rule that fires wins.
addr=':(#?[0−9]+)'
protocol='(https?|ftp|file|gopher|mailto|news|nntp|telnet|wais)'
domain='[a-zA−Z0−9_@]+([.::][a-zA−Z0−9_@]+)*/?[a-zA−Z0−9_?%,#~/−\]+?'
file='([::][a-zA−Z0−9_?%,#~/−\]+)+'

# image files go to page
type is text
data matches '^[a-zA−Z0−9_−/]+$'
data matches '^[a-zA−Z0−9_−/]+$.(jpe?g|gif|bit)'
arg isfile $0
plumb to image
plumb start page −w $file

# URLs go to web browser
type is text
data matches $protocol://$domain$file
plumb to web
plumb start window webbrowser $0

# existing files, possibly tagged by line number, go to edit/sam
type is text
data matches '^[a-zA−Z0−9_−/]+$('
arg isfile $1
data set $file
attr add addr=$3
plumb to edit
plumb start window sam $file

# .h files are looked up in /sys/include and passed to edit/sam
type is text
data matches '^[a-zA−Z0−9]+\.h('$addr')?'
arg isfile /sys/include/$1
data set $file
attr add addr=$3
plumb to edit
plumb start window sam $file

The following simple plumbing rules file is a good beginning set of rules.
# to update: cp /usr/$user/lib/plumbing /mnt/plumb/rules
editor = acme
# or editor = sam
include basic
FILES
/usr/$user/lib/plumbing default rules file.
/mnt/plumb mount point for plumber(4).
/sys/lib/plumb directory for include files.
/sys/lib/plumb/fileaddr public macro definitions.
/sys/lib/plumb/basic basic rule set.

SEE ALSO
plumb(1), plumb(2), plumber(4), regexp(6)
NAME
regexp - regular expression notation

DESCRIPTION
A regular expression specifies a set of strings of characters. A member of this set of strings is said to be matched by the regular expression. In many applications a delimiter character, commonly /, bounds a regular expression. In the following specification for regular expressions the word ‘character’ means any character (rune) but newline.

The syntax for a regular expression e0 is

\[\begin{align*}
e3 & : \text{literal} \mid \text{charclass} \mid .' \mid '^' \mid '$' \mid '(' e0 ')'
e2 & : e3
| e2 \text{ REP}
\text{REP: } '^' \mid '+' \mid '?'
e1 & : e2
| e1 e2
\text{e0: } e1
| e0 ']' e1
\end{align*}\]

A literal is any non-metacharacter, or a metacharacter (one of \.*+?[]()|\$
), or the delimiter preceded by \.

A charclass is a nonempty string s bracketed \[ s \] (or \[^s\]); it matches any character in (or not in) s. A negated character class never matches newline. A substring \(a−b\), with \(a\) and \(b\) in ascending order, stands for the inclusive range of characters between \(a\) and \(b\). In s, the metacharacters –, ], an initial ^, and the regular expression delimiter must be preceded by a \:\; other metacharacters have no special meaning and may appear unescaped.

A \ matches any character.
A \^ matches the beginning of a line; $ matches the end of the line.

The REP operators match zero or more (\*), one or more (+), zero or one (?), instances respectively of the preceding regular expression e2.

A concatenated regular expression, e1 e2, matches a match to e1 followed by a match to e2.

An alternative regular expression, e0 | e1, matches either a match to e0 or a match to e1.

A match to any part of a regular expression extends as far as possible without preventing a match to the remainder of the regular expression.

SEE ALSO
\text{awk(1), ed(1), grep(1), sam(1), sed(1), regexp(2)}}
NAME
rewrite – mail rewrite rules

SYNOPSIS
/mail/lib/rewrite

DESCRIPTION
Mail(1) uses rewrite rules to convert mail destinations into commands used to dispose of the mail. Each line of the file is a rule. Blank lines and lines beginning with # are ignored.

Each rewriting rule consists of (up to) 4 strings:

pattern A regular expression in the style of regexp(6). The pattern is applied to mail destination addresses. The pattern match is case-insensitive and must match the entire address.

type The type of rule; see below.

arg1 An ed(1) style replacement string, with \n standing for the text matched by the nth parenthesized subpattern.

arg2 Another ed(1) style replacement string.

In each of these fields the substring \s is replaced by the login id of the sender and the substring \l is replaced by the name of the local machine.

When delivering a message, mail starts with the first rule and continues down the list until a pattern matches the destination address. It then performs one of the following actions depending on the type of the rule:

>> Append the mail to the file indicated by expanding arg1, provided that file appears to be a valid mailbox.

| Pipe the mail through the command formed from concatenating the expanded arg1 and arg2.

alias Replace the address by the address(es) specified by expanding arg1 and recur.

translate Replace the address by the address(es) output by the command formed by expanding arg1 and recur.

Mail expands the addresses recursively until each address has matched a >> or | rule or until the recursion depth indicates a rewriting loop (currently 32).

If mail(1) is called with more than one address and several addresses match | rules and result in the same expanded arg1, the message is delivered to all those addresses by a single command, composed by concatenating the common expanded arg1 and each expanded arg2. This mail bundling is performed to reduce the number of times the same message is transmitted across a network. For example, with the following rewrite rule

(([^!].*).bell-labs.com)!(.*) | "'/mail/lib/qmail 's 'net\l'' '\2''

if user presotto runs the command

% mail plan9.bell-labs.com!ken plan9.bell-labs.com!rob

there will follow only one execution of the command

/mail/lib/qmail presotto net!plan9.bell-labs.com ken rob

Here /mail/lib/qmail is an rc(1) script used for locally queuing remote mail.

In the event of an error, the disposition of the mail depends on the name of the command executing the rewrite. If the command is called mail and is run by $user, the command will print an error and deposit the message in /mail/box/$user/dead.letter. If the command is called rmail, usually because it was invoked to deliver mail arriving over the network, the message will be returned to the sender. The returned message will appear to have been sent by user postmaster.

SEE ALSO
mail(1)
NAME
smtpd - SMTP listener configuration

DESCRIPTION
The SMTP daemon of *mail*(1) implements the slave side of the SMTP protocol to accept incoming mail on TCP port 25. In general, *smtpd’s* default parameters are sufficient for internal systems on protected networks, but external or gateway systems require additional security mechanisms. The files */mail/lib/smtpd.conf*, containing configuration parameters, and */mail/lib/blocked*, containing banished addresses, provide the means to exercise these facilities.

Input Format
In both files input lines consist of a verb followed by one or more parameters. These tokens are separated by white space or commas and all characters following a # are comments. A # cannot be escaped. Continuation lines are not supported, but verbs that take multiple parameters can be restated on many lines and the associated parameters accumulate into a single set. All token processing is case-insensitive.

Many parameters are *addresses*, either numeric IP addresses in CIDR notation or a *sender address* in UUCP-style format.

An IP address in CIDR notation has the form

    aaa.bbb.ccc.ddd/mask

consisting of a four octet IP address, a slash, and a *mask length* specifying the number of significant high-order bits. The lower the mask length, the larger the range of addresses covered by the CIDR address; see RFC 1878 for a discussion of mask lengths. Missing low-order octets are assumed to be zero. If a mask length is not given, a mask length of 16, 24, or 32 is assumed for addresses containing two, three, or four octets, respectively. These mask lengths select a class B, class C or Class D address block. Notice that this convention differs from the standard treatment, where the default mask length depends on the allocation class of the network block containing the address.

*Sender addresses* are specified in UUCP notation as follows:

    [domain!]...domain!user

It is seldom necessary to specify more than one domain. When *domain* is missing or *, the address selects the specified user in all domains. A *domain* of the form *.* domain selects the domain and all of its sub-domains. For example, example.com!user only matches the account *user* in domain *example.com*, while *.* example.com!user selects that account in example.com and all of its sub-domains. When *user* is omitted or *, the address selects all users in the specified domain. Finally, when * is the last character of the user name it is a wildcard matching all user names beginning with *user*. This limited pattern matching capability should be used with care. For safety, the sender addresses *+, !, +!, !* and *! are ignored.

/mail/lib/smtpd.conf
This file contains configuration options and parameters describing the local domain. Many of the options can also be specified on the command line; command line options always override the values in this file. Configuration options are:

defaultdomain *domain*
   The name of the local domain; it is appended to addresses lacking a domain qualification. This is identical to the –h command line option.

norelay [on|off]
   If on is specified, relaying is prohibited from unauthorized networks to external domains. Authorized networks and domains must be specified by the *ournets* and *ourdomains* verbs described below. Setting this option on is equivalent to specifying the –f command line flag, but the list of networks and domains can only be specified in this file.

verifysenderdomain [on|off]
   When on, *smtpd* verifies that the first domain of the sender’s address exists. The test is cursory; it checks only that there is a DNS delegation for the domain. Setting the option on is equivalent to specifying the –z command line option and is useful for detecting
some unreturnable messages as well as messages with randomly generated domain names.

`saveblockedmsg [on|off]`
When on, causes copies of blocked messages to be saved in subdirectories of `/mail/queue.dum`. Directories are named with the date and file names are random numbers. If this option is off blocked messages are discarded. Setting this option on is equivalent to specifying the –s command line option.

`ournets IP address [, IP address, ..., IP address]`
This option specifies trusted source networks that are allowed to relay mail to external domains. These are usually the internal networks of the local domain, but they can also include friendly external networks. Addresses are in CIDR notation.

`ourdomains domain [, domain, ..., domain]`
This option specifies destination domains that are allowed to receive relayed mail. These are usually the domains served by a gateway system. Domain specifications conform to the format for sender addresses given above.

When the norelay option is enabled or the –f command line option given, relaying is allowed only if the source IP address is in `ournets` or the destination domain is specified in `ourdomains`.

**Blocked Addresses**

`Smtpd` consults `/mail/ratify` (see `ratfs(4)`) for a list of banned addresses. Messages received from these addresses are rejected with a 5xx-series SMTP error code. There is no option to turn blocking on or off; if `/mail/ratify` is mounted, `smtpd` will use it, even for connections from trusted networks.

The command line format and address specifications conform to the notation described above. If the parameters of the verb is sender addresses in UUCP format, the line must begin with an `*` character; if the parameters are one or more IP addresses, the `*` must precede the verb. Most verbs cause messages to be rejected; verbs of this class generally select different error messages. The remaining verbs specify addresses that are always accepted, in effect overriding blocked addresses. The file is processed in order, so an override must precede its associated blocked address. Supported verbs are:

- `dial IP address [, ..., IP address]`
The parameters are IP addresses associated with dial-up ports. The rejection message states that connections from dial-up ports are not accepted. Copies of messages are never saved.

- `block address [, ..., address]`
Messages from addresses matching the parameters are rejected with an error message saying that spam is not accepted. The message is saved if the option is enabled.

- `relay address [, ..., address]`
This verb is identical to `block`, but the error message states that the message is rejected because the sending system is being used as a spam relay.

- `deny address [, ..., address]`
The deny command rejects a message when the sender address matches one of its parameters. The rejection message asks the sender to contact postmaster®
`hostdomain` for further information. This verb is usually used to block inadvertently abusive traffic, for example, mail loops and stuck senders. Messages are never saved.

- `allow address [, ..., address]`
The allow verb negates the effect of subsequent blocking commands. It is useful when a large range of addresses contains a few legitimate addresses, for example, when a mail server is in a Class C network block of modem ports. Rather than enumerate the dial ports, it is easier to block the entire Class C with a dial command, and precede it with an override for the address of the mail server. Similarly, it is possible to block mail from an entire domain while accepting mail from a few friendly senders in the domain. The verb accept is a synonym for allow.

`Scanmail(8)` describes spam detection software that works well with the capabilities described here and `mail(1)` defines additional `smtpd` command line arguments applicable to exposed systems.

**SEE ALSO**

`mail(1), ratfs(4), scanmail(8)`
NAME
snap - process snapshots

DESCRIPTION
Process snapshots are used to save a process image for debugging on another machine or at another time. They are like old Unix core dumps but can hold multiple process images and are smaller.

The first line of a snapshot begins with the prefix “process snapshot” and often contains other information as well, such as creation time, user name, system name, cpu type, and kernel type. This information is intended for humans, not programs. Programs reading snapshots should only check that this line begins with the specified prefix.

Throughout the rest of the snapshot, decimal strings are always right-justified, blank-padded to at least 11 characters, and followed by a single space character.

The rest of the snapshot is one or more records, each of which begins with a one-line header. This header is a decimal process id followed by an identification string, which denotes the type of data in the record.

Records of type fd, fpregs, kregs, noteid, ns, proc, regs, segment, and status are all formatted as a decimal number $n$ followed by $n$ bytes of data. This data is the contents of the file of the same name found in /proc.

The format of the mem and text sections is not as simple. These sections contain one or more page descriptions. Each describes a one kilobyte page of data. If the section is not a multiple of a kilobyte in size, the last page will be shorter. Each description begins with a one-byte flag. If the flag is r, then it is followed by a page of binary data. If the flag is z, then the data is understood to be zeros, and is omitted. If the flag is m or t, then it is followed by two decimal strings $p$ and $o$, indicating that this page is the same as the page at offset $o$ of the memory or text segment for process $p$. This data must have been previously described in the snapshot, and the offset must be a multiple of a kilobyte.

It is not guaranteed that any of the sections described above be in a process snapshot, although the snapshot quickly becomes useless when too much is missing.

Memory and text images may be incomplete. The memory or text file for a given process may be split across multiple disjoint sections in the snapshot.

SEE ALSO
proc(3), snap(4).
NAME
style – Plan 9 coding conventions for C

DESCRIPTION
Plan 9 C code has its own conventions. You would do well to follow them. Here are a few:
- don’t use // comments; some old Plan 9 code does, but we’re converting it as we touch it. We
do sometimes use // to comment-out a few lines of code.
- avoid gotos.
- no tabs expanded to spaces.
- surround a binary operator (particular a low precedence one) with spaces; don’t try to write the
most compact code possible but rather the most readable.
- parenthesize expressions involving arithmetic and bit-wise operators; otherwise don’t paren-
thesize heavily (e.g., as in Pascal).
- no white space before opening braces.
- no white space after the keywords if, for, while, etc.
- no braces around single–line blocks (e.g., if, for, and while bodies).
- integer–valued functions return –1 on error, 0 or positive on success.
- functions that return errors should set errstr(2).
- variable and function names are all lowercase, with no underscores.
- enum or #defined constants should be Uppercase (or UPPERCASE).
- struct tags are Uppercase, with matching typedefs.
- automatic variables (local variables inside a function) are never initialized at declaration.
- follow the standard idioms: use x < 0 not 0 > x, etc.
- don’t write !strcmp (nor !memcmp, etc.) nor if(memcmp(a, b, c)); always explicitly
compare the result of string or memory comparison with zero using a relational operator.
Ultimately, the goal is to write code that fits in with the other code around it and the system as a
whole. If the file you are editing already deviates from these guidelines, do what it does. After you
edit a file, a reader should not be able to tell just from coding style which parts you worked on.

COMMENTS
If your code is readable, you shouldn’t need many comments. A line or two comment above a
function explaining what it does is always welcome.
Comment any code you find yourself wondering about for more than 2 seconds, even if it’s to say
that you don’t understand what’s going on. Explain why.
Don’t use commenting as an excuse for writing confusing code. Rewrite the code to make it clear.

EFFICIENCY
Do the simple thing. Don’t optimize unless you’ve measured the code and it is too slow. Fix the
data structures and the algorithms instead of going for little 5% tunings.

SEE ALSO
“Notes on Programming in C”, Rob Pike,

BUGS
Some programs use very different styles, for example, rc.
Some programs and programmers diverge from the above rules due to habits formed long before
these rules. Notably, some programs have a single space after a keyword and before an opening
brace, and some initialize automatic variables at declaration.
NAME
thumbprint - public key thumbprints

DESCRIPTION
Applications in Plan 9 that use public keys for authentication, for example by calling tlsClient and okThumbprint (see pushtls(2)), check the remote side's public key by comparing against thumbprints from a trusted list. The list is maintained by people who set local policies about which servers can be trusted for which applications, thereby playing the role taken by certificate authorities in PKI-based systems. By convention, these lists are stored as files in /sys/lib/tls/ and protected by normal file system permissions.

Such a thumbprint file comprises lines made up of attribute/value pairs of the form attr=value or attr. The first attribute must be x509 and the second must be sha1=\{hexchecksum of binary certificate\}. All other attributes are treated as comments. The file may also contain lines of the form #include file

For example, a web server might have thumbprint
x509 sha1=8fe472d31b360a8303cd29f92bd734813cbd923c cn=*.cs.bell-labs.com

SEE ALSO
pushtls(2)
NAME
users – file server user list format

DESCRIPTION
The permanent file servers each maintain a private list of users and groups, in /adm/users by
convention. Each line in the file has the format

id : name : leader : members

where name and leader are printable strings excluding the characters ?, =, +, -, /, and :, and
members is a comma-separated list of such strings. Such a line defines a user and a group with
the given name; the group has a group leader given by leader and group members given by the
user names in members. The leader field may be empty, in which case any group member is a
group leader. The members field may be empty.

Lines beginning with # are ignored.

The id in a line is an identifier used in the on-disk structures maintained by a file server; there
should be no duplicate ids in the file. In fossil(4), ids are arbitrary text strings, typically the same
as name. In older Plan 9 file servers, ids are small decimal numbers. In those, a negative id is spe-
cial: a user with a negative id cannot attach to the file server. The file /adm/userss itself is
owned by user adm and write protected to others, so it can only be changed via console com-
mands.

SEE ALSO
intro(5), stat(5), fossilcons(8)
NAME
UTF, Unicode, ASCII, rune - character set and format

DESCRIPTION
The Plan 9 character set and representation are based on the Unicode Standard and on the ISO
multibyte UTF-8 encoding (Universal Character Set Transformation Format, 8 bits wide). The Uni-
code Standard represents its characters in 16 bits; UTF-8 represents such values in an 8-bit byte
stream. Throughout this manual, UTF-8 is shortened to UTF.

In Plan 9, a rune is a 16-bit quantity representing a Unicode character. Internally, programs may
store characters as runes. However, any external manifestation of textual information, in files or
at the interface between programs, uses a machine-independent, byte-stream encoding called UTF.

UTF is designed so the 7-bit ASCII set (values hexadecimal 00 to 7F), appear only as themselves in
the encoding. Runes with values above 7F appear as sequences of two or more bytes with values
only from 80 to FF.

The UTF encoding of the Unicode Standard is backward compatible with ASCII: programs presented
only with ASCII work on Plan 9 even if not written to deal with UTF, as do programs that deal with
uninterpreted byte streams. However, programs that perform semantic processing on ASCII
graphic characters must convert from UTF to runes in order to work properly with non-ASCII input.
See rune(2).

Letting numbers be binary, a rune x is converted to a multibyte UTF sequence as follows:

01. x in [00000000.0bbbbbbb] → 0bbbbbbb
10. x in [00000bbb.bbbbbbbb] → 110bbbbbb, 1bbbbbbb
11. x in [bbbbbbbb.bbbbbbbb] → 1110bbbbbb, 1bbbbbbb, 1bbbbbbb

Conversion 01 provides a one-byte sequence that spans the ASCII character set in a compatible
way. Conversions 10 and 11 represent higher-valued characters as sequences of two or three
bytes with the high bit set. Plan 9 does not support the 4, 5, and 6 byte sequences proposed by
X-Open. When there are multiple ways to encode a value, for example rune 0, the shortest encod-
ing is used.

In the inverse mapping, any sequence except those described above is incorrect and is converted
to rune hexadecimal FFFD.

FILES
/lib/unicodem  table of characters and descriptions, suitable for look(1).

SEE ALSO
ascii(1), tcs(1), rune(2), keyboard(6), The Unicode Standard.
NAME
venti – archival storage server

DESCRIPTION
Venti is a block storage server intended for archival data. In a Venti server, the SHA1 hash of a block’s contents acts as the block identifier for read and write operations. This approach enforces a write-once policy, preventing accidental or malicious destruction of data. In addition, duplicate copies of a block are coalesced, reducing the consumption of storage and simplifying the implementation of clients.

This manual page documents the basic concepts of block storage using Venti as well as the Venti network protocol.

Venti(1) documents some simple clients. Vac(1) and vacfs(4) are more complex clients.

Venti(2) describes a C library interface for accessing Venti servers and manipulating Venti data structures.

Venti(8) describes the programs used to run a Venti server.

Scores
The SHA1 hash that identifies a block is called its score. The score of the zero-length block is called the zero score.

Scores may have an optional label: prefix, typically used to describe the format of the data. For example, vac(1) uses a vac: prefix, while vbackup(8) uses prefixes corresponding to the file system types: ext2:, ffs:, and so on.

Files and Directories
Venti accepts blocks up to 56 kilobytes in size. By convention, Venti clients use hash trees of blocks to represent arbitrary-size data files. The data to be stored is split into fixed-size blocks and written to the server, producing a list of scores. The resulting list of scores is split into fixed-size pointer blocks (using only an integer number of scores per block) and written to the server, producing a smaller list of scores. The process continues, eventually ending with the score for the hash tree’s top-most block. Each file stored this way is summarized by a VtEntry structure recording the top-most score, the depth of the tree, the data block size, and the pointer block size. One or more VtEntry structures can be concatenated and stored as a special file called a directory. In this manner, arbitrary trees of files can be constructed and stored.

Scores passed between programs conventionally refer to VtRoot blocks, which contain descriptive information as well as the score of a directory block containing a small number of directory entries.

Conventionally, programs do not mix data and directory entries in the same file. Instead, they keep two separate files, one with directory entries and one with metadata referencing those entries by position. Keeping this parallel representation is a minor annoyance but makes it possible for general programs like venti/copy (see venti(1)) to traverse the block tree without knowing the specific details of any particular program’s data.

Block Types
To allow programs to traverse these structures without needing to understand their higher-level meanings, Venti tags each block with a type. The types are:

- VtDataType 000 data
- VtDataType+1 001 scores of VtDataType blocks
- VtDataType+2 002 scores of VtDataType+1 blocks
- VtDirType 010 VtEntry structures
- VtDirType+1 011 scores of VtDirType blocks
- VtDirType+2 012 scores of VtDirType+1 blocks
- VtRootType 020 VtRoot structure

The octal numbers listed are the type numbers used by the commands below. (For historical reasons, the type numbers used on disk and on the wire are different from the above. They do not distinguish VtDataType+n blocks from VtDirType+n blocks.)
Zero Truncation

To avoid storing the same short data blocks padded with differing numbers of zeros, Venti clients working with fixed-size blocks conventionally ‘zero truncate’ the blocks before writing them to the server. For example, if a 1024-byte data block contains the 11-byte string ‘hello world’ followed by 1013 zero bytes, a client would store only the 11-byte block. When the client later read the block from the server, it would append zero bytes to the end as necessary to reach the expected size.

When truncating pointer blocks (VtDataType+n and VtDirType+n blocks), trailing zero scores are removed instead of trailing zero bytes.

Because of the truncation convention, any file consisting entirely of zero bytes, no matter what its length, will be represented by the zero score: the data blocks contain all zeros and are thus truncated to the empty block, and the pointer blocks contain all zero scores and are thus also truncated to the empty block, and so on up the hash tree.

Network Protocol

A Venti session begins when a client connects to the network address served by a Venti server; the conventional address is tcp!server!venti (the venti port is 17034). Both client and server begin by sending a version string of the form venti-versions-comment\n. The versions field is a list of acceptable versions separated by colons. The protocol described here is version 02. The client is responsible for choosing a common version and sending it in the VtThello message, described below.

After the initial version exchange, the client transmits requests (T-messages) to the server, which subsequently returns replies (R-messages) to the client. The combined act of transmitting (receiving) a request of a particular type, and receiving (transmitting) its reply is called a transaction of that type.

Each message consists of a sequence of bytes. Two–byte fields hold unsigned integers represented in big–endian order (most significant byte first). Data items of variable lengths are represented by a one–byte field specifying a count, n, followed by n bytes of data. Text strings are represented similarly, using a two–byte count with the text itself stored as a UTF–encoded sequence of Unicode characters (see utf(6)). Text strings are not NUL–terminated: n counts the bytes of UTF data, which include no final zero byte. The NUL character is illegal in text strings in the Venti protocol. The maximum string length in Venti is 1024 bytes.

Each Venti message begins with a two–byte size field specifying the length in bytes of the message, not including the length field itself. The next byte is the message type, one of the constants in the enumeration in the include file <venti.h>. The next byte is an identifying tag, used to match responses to requests. The remaining bytes are parameters of different sizes. In the message descriptions, the number of bytes in a field is given after the field name. The notation parameter[n] where n is not a constant represents a variable–length parameter: n[1] followed by n bytes of data forming the parameter. The notation string[s] (using a literal s character) is shorthand for s[2] followed by s bytes of UTF–8 text. The notation parameter[] where parameter is the last field in the message represents a variable–length field that comprises all remaining bytes in the message.

All Venti RPC messages are prefixed with a field size[2] giving the length of the message that follows (not including the size field itself). The message bodies are:

VtPping tag[1]
VtRping tag[1]
VtRead tag[1] data[]
VtRwrite tag[1] score[20]
VtTsync tag[1]
VtRsync tag[1]
VtRerror tag[], error[]
VtTgoodbye tag[]

Each T-message has a one-byte tag field, chosen and used by the client to identify the message. The server will echo the request’s tag field in the reply. Clients should arrange that no two outstanding messages have the same tag field so that responses can be distinguished.

The type of an R-message will either be one greater than the type of the corresponding T-message or Rerror, indicating that the request failed. In the latter case, the error field contains a string describing the reason for failure.

Venti connections must begin with a hello transaction. The VtThello message contains the protocol version that the client has chosen to use. The fields strength, crypto, and codec could be used to add authentication, encryption, and compression to the Venti session but are currently ignored. The rcrypto, and rcodec fields in the VtRhello response are similarly ignored. The uid and sid fields are intended to be the identity of the client and server but, given the lack of authentication, should be treated only as advisory. The initial hello should be the only hello transaction during the session.

The ping message has no effect and is used mainly for debugging. Servers should respond immediately to pings.

The read message requests a block with the given score and type. Use vttodisktype and vtfromdisktype (see venti(2)) to convert a block type enumeration value (VtDataType, etc.) to the type used on disk and in the protocol. The count field specifies the maximum expected size of the block. The data in the reply is the block’s contents.

The write message writes a new block of the given type with contents data to the server. The response includes the score to use to read the block, which should be the SHA1 hash of data.

The Venti server may buffer written blocks in memory, waiting until after responding to the write message before writing them to permanent storage. The server will delay the response to a sync message until after all blocks in earlier write messages have been written to permanent storage.

The goodbye message ends a session. There is no VtRgoodbye: upon receiving the VtTgoodbye message, the server terminates the connection.

SEE ALSO
venti(1), venti(2), venti(8)
NAME
venti.conf – a venti configuration file

DESCRIPTION
A venti configuration file enumerates the various index sections and arenas that constitute a venti system. The components are indicated by the name of the file, typically a disk partition, in which they reside. The configuration file is the only location that file names are used. Internally, venti uses the names assigned when the components were formatted with fmtarenas or fmtisect (see venti−fmt(8)). In particular, by changing the configuration a component can be copied to a different file.

The configuration file consists of lines in the form described below. Lines starting with # are comments.

index name
Names the index for the system.
arenas file
File contains a collection of arenas, formatted using fmtarenas.
isect file
File contains an index section, formatted using fmtisect.

After formatting a venti system using fmtindex, the order of arenas and index sections should not be changed. Additional arenas can be appended to the configuration.

The configuration file optionally holds configuration parameters for the venti server itself. These are:
mem cachesize
bcmem blockcachesize
icmem indexcachesize
addr ventiaddress
httpaddr httpaddress
queuewrites

See venti(8) for descriptions of these variables.

EXAMPLE
# a sample venti configuration file
#
# formatted with
# venti/fmtarenas arena. /tmp/disks/arenas
# venti/fmtisect isect0 /tmp/disks/isect0
# venti/fmtisect isect1 /tmp/disks/isect1
# venti/fmtindex venti.conf
#
# server is started with
# venti/venti
#
# the name of the index
index main
#
# the index sections
isect /tmp/disks/isect0
isect /tmp/disks/isect1
#
# the arenas
arenas /tmp/disks/arenas

SEE ALSO
fs(3), venti(8), venti−fmt(8)
NAME

vgadb - VGA controller and monitor database

DESCRIPTION

The VGA database, /lib/vgadb, consists of two parts, the first describing how to identify and program a VGA controller and the second describing the timing parameters for known monitors to be loaded into a VGA controller to give a particular resolution and refresh rate. Conventionally, at system boot, the program aux/vga (see vga(8)) uses the monitor type in /env/monitor, the display resolution in /env/vgasize, and the VGA controller information in the database to find a matching monitor entry and initialize the VGA controller accordingly.

The file comprises multi-line entries made up of attribute/value pairs of the form attr=value or sometimes just attr. Each line starting without white space starts a new entry. Lines starting with # are comments.

The first part of the database, the VGA controller identification and programming information, consists of a number of entries with attribute ctrlr and no value. Within one of these entries the following attributes are meaningful:

- **nnnn** an offset into the VGA BIOS area. The value is a string expected to be found there that will identify the controller. For example, 0xC00068="#9GXE64 Pro" would identify a #9GXEpro VGA controller if the string #9GXE64 Pro was found in the BIOS at address 0xC0068. There may be more than one identifier attribute per controller. If a match cannot be found, the first few bytes of the BIOS are printed to help identify the card and create a controller entry.

- **nnnn−mmmmm** A range of the VGA BIOS area. The value is a string as above, but the entire range is searched for that string. The string must begin at or after nnnn and not contain any characters at or after mmmm. For example, 0xC00000–0xC02000="MACH64LP" identifies a Mach 64 controller with the string MACH64LP occurring anywhere in the first 512 bytes of BIOS memory.

- **ctrlr** VGA controller chip type. This must match one of the VGA controller types known to /dev/vgactl (see vga(3)) and internally to aux/vga. Currently ark2000pv, clgd542x, ct65540, ct65545, cyber938x, et4000, hiqvvideo, ibm8514, mach32, mach64, mach64xx, mga2164w, neomagic, s3801, s3805, s3928, t2r4, trio64, virge, vision864, vision964, vision968, and w30c516 are recognized.

- **ramdac** RAMDAC controller type. This must match one of the types known internally to aux/vga. Currently att20c490, att20c491, att21c492, att21c498, bt485, rgb524mm, sc15025, stg1702, tvp3020, tvp3025, and tvp3026 are recognized.

- **clock** clock generator type. This must match one of the types known internally to aux/vga. Currently ch9294, icd2061a, ics2494, ics2494a, s3clock, tvp3025clock, and tvp3026clock are recognized.

- **hwgc** hardware graphics cursor type. This must match one of the types known to /dev/vgactl and internally to aux/vga. Currently ark200pvhwgc, bt485hwgc, clgd542xhwgc, clgd546xhwgc, ct65545hwgc, cyber938xhwgc, hiqvvideohwgc, mach64xxhwgc, mga2164xhwgc, neomagicxhwgc, rgb524xhwgc, s3hwgc, t2r4xhwgc, tvp3020xhwgc, and tvp3026xhwgc are recognized.

- **membw** Memory bandwidth in megabytes per second. Vga chooses the highest refresh rate possible within the constraints of the monitor (explained below) and the card’s memory bandwidth.

- **linear** Whether the card supports a large (>64kb) linear memory window. The value is either 1 or 0 (equivalent to unspecified). The current kernel graphics subsystem requires a linear window; entries without linear=1 are of historic value only.

- **link** This must match one of the types known internally to aux/vga. Currently vga and ibm8514 are recognized. The type vga handles generic VGA functions and should almost always be included. The type ibm8514 handles basic graphics accelerator initialization on controllers such as the early S3 family of GUI chips.
The clock, ctlr, link, and ramdac values can all take an extension following a '-' that can be used as a speed-grade or subtype; matching is done without the extension. For example, ramdac=stg1702−135 indicates the STG1702 RAMDAC has a maximum clock frequency of 135MHz, and clock=ics2494a−324 indicates that the frequency table numbered 324 should be used for the ICS2494A clock generator.

The functions internal to aux/vga corresponding to the clock, ctlr, link, and ramdac values will be called in the order given for initialization. Sometimes the clock should be set before the RAMDAC is initialized, for example, depending on the components used. In general, link=vga will always be first and, if appropriate, link=ibm8514 will be last.

The entries in the second part of /lib/vgadb have as attribute the name of a monitor type and the value is conventionally a resolution in the form XxY, where X and Y are numbers representing width and height in pixels. The monitor type (i.e. entry) include has special properties, described below and shown in the examples. The remainder of the entry contains timing information for the desired resolution. Within one of these entries the following attributes are meaningful:

- **clock**: the video dot-clock frequency in MHz required for this resolution. The value 25.175 is known internally to vga(8) as the baseline VGA clock rate.
- **defaultclock**: the default video dot-clock frequency in MHz used for this resolution when no memory bandwidth is specified for the card or when vga cannot determine the maximum clock frequency of the card.
- **shb**: start horizontal blanking, in character clocks.
- **ehb**: end horizontal blanking, in character clocks.
- **ht**: horizontal total, in character clocks.
- **vrs**: vertical refresh start, in character clocks.
- **vre**: vertical refresh end, in character clocks.
- **vt**: vertical total, in character clocks.
- **hsync**: horizontal sync polarity. Value must be + or −.
- **vsync**: vertical sync polarity. Value must be + or −.
- **interlace**: interlaced mode. Only value v is recognized.
- **alias**: continue, replacing the alias line by the contents of the entry whose attribute is given as value.
- **include**: continue, replacing this include line by the contents of the previously defined include monitor type with matching value. (See the examples.) Any non-zero attributes already set will not be overwritten. This is used to save duplication of timing information. Note that value is not parsed, it is only used as a string to identify the previous include= value monitor type entry.

The values given for shb, ehb, ht, vrs, vre, vt, hsync, and vsync are beyond the scope of this manual page. See the book by Ferraro for details.

**EXAMPLES**

Basic ctlr entry for a laptop with a Chips and Technology 65550 controller:

```
ctlr # NEC Versa 6030X/6200MX
  0xC0090="CHIPS 65550 PCI & VL Accelerated VGA BIOS"
  link=vga
  ctlr=hiqvideo linear=1
  hwgc=hiqvideo
```

A more complex entry. Note the extensions on the clock, ctlr, and ramdac attributes. The order here is important: the RAMDAC clock input must be initialized before the RAMDAC itself. The clock frequency is selected by the ET4000 chip.

```
ctlr # Hercules Dynamite Power
  0xC0076="Tseng Laboratories, Inc. 03/04/94 V8.00N"
  link=vga
  clock=ics2494a−324
  ctlr=et4000−w32p
  ramdac=stg1702−135
```

Monitor entry for type vga (the default monitor type used by vga(8)) and resolution 640x480x[18].

```
include = 640x480@60Hz # 60Hz, 31.5KHz
  clock=25.175
```

803
shb=664 ehb=760 ht=800
vrs=491 vre=493 vt=525

vga = 640x480   # 60Hz, 31.5KHz
include=640x480@60Hz

Entries for multisync monitors with video bandwidth up to 65MHz.
#
# Multisync monitors with video bandwidth up to 65MHz.
#
multisync65 = 1024x768   # 60Hz, 48.4KHz
include=1024x768@60Hz

multisync65 = 1024x768i   # 87Hz, 35.5KHz (interlaced)
include=1024x768i@87Hz

multisync65
alias=vga

Note how this builds on the existing vga entries.

FILES
/lib/vgadb

SEE ALSO
ndb(2), vga(3), ndb(6), 9load(8), vga(8)
Richard E. Ferraro, Programming Guide to the EGA, VGA and Super VGA Cards, Third Edition

BUGS
The database should provide a way to use the PCI bus as well as BIOS memory to identify cards.

ADDITION OF A NEW MONITOR
Adding a new monitor is usually fairly straightforward, as most modern monitors are multisync
and the only interesting parameter is the maximum video bandwidth. Once the timing parameters
are worked out for a particular maximum video bandwidth as in the example above, an entry for a
new monitor with that limit is simply
#
# Sony CPD–1304
# Horizontal timing:
#   Allowable frequency range: 28–50KHz
# Vertical timing:
#   Allowable frequency range: 50–87Hz
#
cpd–1304
alias=multisync65

Even this is not necessary, as the monitor type could simply be given as multisync65.

ADDITION OF A NEW VGA CONTROLLER
While the use of this database formalizes the steps needed to program a VGA controller, unless
you are lucky and all the important components on a new VGA controller card are interconnected
in the same way as an existing entry, adding a new entry requires adding new internal types to
vga(8). Fortunately, the unit of variety has, for the most part, shifted from individual components
to entire video chipsets. Thus in lucky cases all that is necessary is the addition of another
0xNMMMM= line to the entry for the controller. This is particularly true in the case of the ATI Mach
64 and the S3 Virge.

If you need to actually add support for a controller with a different chipset, you will need the data
sheets for the VGA controller as well as any RAMDAC or clock generator (these are commonly inte-
grated into the controller). You will also need to know how these components interact. For example,
a common combination is an S3 86C928 VGA chip with an ICD2061A clock generator. The
ICD2061A is usually loaded by clocking a serial bit-stream out of one of the 86C928 registers.
Similarly, the RAMDAC may have an internal clock-doubler and/or pixel–multiplexing modes, in
which case both the clock generator and VGA chip must be programmed accordingly. Hardware
acceleration for rectangle fills and block copies is provided in the kernel; writing code to handle
this is necessary to achieve reasonable performance at high pixel depths.
NAME
intro - introduction to databases

DESCRIPTION
This manual section describes databases available on Plan 9 and the commands that access them. Some of them involve proprietary data that is not distributed outside Bell Laboratories.
NAME
astro - print astronomical information

SYNOPSIS
astro [-dlpsatokm] [-c n] [-C d] [-e obj1 obj2]

DESCRIPTION
 Astro reports upcoming celestial events, by default for 24 hours starting now. The options are:

d  Read the starting date. A prompt gives the input format.

l  Read the north latitude, west longitude, and elevation of the observation point. A prompt
gives the input format. If l is missing, the initial position is read from the file
/lib/sky/herel.

C  Report for n (default 1) successive days.

C  Used with -c, set the interval to d days (or fractions of days).

C  Report distance between the centers of objects, in arc seconds, during eclipses or occultations
involving obj1 and obj2.

P  Print the positions of objects at the given time rather than searching for interesting con-
njunctions. For each, the name is followed by the right ascension (hours, minutes, seconds),
declination (degrees, minutes, seconds), azimuth (degrees), elevation (degrees), and
semidiameter (arc seconds). For the sun and moon, the magnitude is also printed. The
first line of output presents the date and time, sidereal time, and the latitude, longitude,
and elevation.

S  Print output in English words suitable for speech synthesizers.

A  Include a list of artificial earth satellites for interesting events. (There are no orbital ele-
ments for the satellites, so this option is not usable.)

T  Read \( \Delta T \) from standard input. \( \Delta T \) is the difference between ephemeris and universal time
(seconds) due to the slowing of the earth’s rotation. \( \Delta T \) is normally calculated from an
empirical formula. This option is needed only for very accurate timing of occultations,
eclipses, etc.

O  Search for stellar occultations.

K  Print times in local time (‘kitchen clock’) as described in the timezone environment vari-
able.

M  Includes a single comet in the list of objects. This is modified (in the source) to refer to an
approaching comet but in steady state usually refers to the last interesting comet (currently
Hale-Bopp, C/1995 O1).

FILES
/lib/sky/estartab  ecliptic star data
/lib/sky/herel    default latitude (N), longitude (W), and elevation (meters)

SOURCE
/sys/src/cmd/astro

SEE ALSO
scat(7)

BUGS
 The k option reverts to GMT outside of 1970–2036.
NAME
 audio - digital audio jukebox

SYNOPSIS
 audio

DESCRIPTION
 Audio is a shell script that binds in the lib/audio directory from appropriate places and then execs games/agui, which is a digital audio jukebox. This program is a GUI that can walk through a database of songs. If the GUI is asked to play a song, it runs games/pac4dec which decompresses PAC audio files in real time. For this to work, the computer must be equipped with an audio device, it must be fast enough to do the conversion (about 50 Mips) and the bandwidth to the converter must be on the order of 20Kbs.

Mouse buttons
 Button 1 is used to walk around the menu and exercise options. Pointing at a line of text and clicking button 1 will isolate that line and expand any structure below it.
 Button 2 queues songs to be played. Pointing and clicking button 2 on a song will play that song. Clicking on a collection of songs will play all of the songs in the collection.
 Button 3 saves songs in a list for later recall. Pointing and clicking button 3 on a song or a collection of songs will append the identities of those songs to the file $home/tmp/agui/def.

Keyboard
 Typing a word at the keyboard will search for all songs that contain that word in either the artist field or the title field. Typing more that one word will search for songs containing all the words. A word may end in * and will match all words starting with the string before the *. A word starting with . will fetch a list of songs from the file $home/tmp/agui/word. Files of this form may be created with button 3 described above. This mechanism is used to define private play lists.

FILES
 /lib/audio/rawmap database description.
 /lib/audio/... audio files.

SOURCE
 /sys/src/games/agui the graphical user interface.
 /sys/src/games/pac4dec PAC audio format decoder.
 /sys/src/games/pacenc PAC audio format encoder.

CAVEAT
 Bell Labs Licensing has OKed the jukebox to expand from a private toy to a communal research tool. Under fair use we can pool all our CDs and put them in the library. Any use larger than this, especially any public demonstrations, should seek a performance license. The current (Oct 1999) database contains 30,000 songs totaling 100 Gbytes.
NAME
dict - dictionary browser

SYNOPSIS
dict [−k][−d dictname][−c command][pattern]

DESCRIPTION
Dict is a dictionary browser. If a pattern is given on the command line, dict prints all matching entries; otherwise it repeatedly accepts and executes commands. The options are
−d dictname Use the given dictionary. The default is oed, the second edition of the Oxford English Dictionary. A list of available dictionaries is printed by option −d?.
−c command Execute one command and quit. The command syntax is described below.
−k Print a pronunciation key.

Patterns are regular expressions (see regexp(6)), with an implicit leading ^ and trailing $. Patterns are matched against an index of headwords and variants, to form a ‘match set’. By default, both patterns and the index are folded: upper case characters are mapped into their lower case equivalents, and Latin accented characters are mapped into their non–accented equivalents. In interactive mode, there is always a ‘current match set’ and a ‘current entry’ within the match set. Commands can change either or both, as well as print the entries or information about them.

Commands have an address followed by a command letter. Addresses have the form:
/re/ Set the match set to all entries matching the regular expression re, sorted in dictionary order. Set the current entry to the first of the match set.
!re! Like /re/ but use exact matching, i.e., without case and accent folding.
n An integer n means change the current entry to the nth of the current match set.
#n The integer n is an absolute byte offset into the raw dictionary. (See the A command, below.)
addr+ After setting the match set and current entry according to addr, change the match set and current entry to be the next entry in the dictionary (not necessarily in the match set) after the current entry.
addr− Like addr+ but go to previous dictionary entry.

The command letters come in pairs: a lower case and the corresponding upper case letter. The lower case version prints something about the current entry only, and advances the current entry to the next in the match set (wrapping around to the beginning after the last). The upper case version prints something about all of the match set and resets the current entry to the beginning of the set.
p,P Print the whole entry.
h,H Print only the headword(s) of the entry.
a,A Print the dictionary byte offset of the entry.
r,R Print the whole entry in raw format (without translating special characters, etc.).

If no command letter is given for the first command, H is assumed. After an H, the default command is p. Otherwise, the default command is the previous command.

FILES
/lib/dict/oed2
/lib/dict/oed2index
Other files in /lib.

SEE ALSO
regexp(6)

SOURCE
/sys/src/cmd/dict

BUGS
A font with wide coverage of the Unicode Standard should be used for best results. (Try /lib/font/bit/pelm/unicode.9.font.)
If the pattern doesn’t begin with a few literal characters, matching takes a long time.
The dictionaries are not distributed outside Bell Labs.
NAME
juke - music jukebox

SYNOPSIS
juke [-t] [-w] [-h srvhost] [-s srvname]
games/jukebox [-t] [-w]
games/jukefs [-m mountpoint] [-s srvname] [mapfile]

DESCRIPTION
Jukebox controls a playlist server (see playlists(7)) through a graphical user interface. It connects
to a music database server which reads a set of map files that describe recordings and their loca-
tion. Currently, there is one set of maps, mostly for classical music, with some jazz and other stuff
thrown in. These are served by jukefs, which presents a file system conventionally mounted at
/mnt/juke. The playlist, explained below, is managed by a file system implemented by
playlists(7) and normally mounted on /mnt.

Jukebox is most easily started through the juke shell script.

Jukebox has four windows, which can be selected by clicking the appropriate tab at the top of the
window.

Above the tab are nine buttons and a volume slider. The buttons, shown below, are named, from
left to right, Exit, Pause, Play, Halt, Back, Forward, Root, Delete, and Help. The buttons are active
when they are displayed in dark green (or red). When they are pale blue they are inactive. The Exit
button is always active; it exits the program (but leaves the playlist and music database servers
running).

The browse window is for browsing through the music and selecting music to play. Browsing down
in the music hierarchy is done by clicking button one on an item. Clicking button three goes back up.
Clicking button two recursively adds all files below the selected item to the playlist.

The selected music is displayed in the playlist window. The track currently playing is shown in the
playing window.

The Root button browses back to the root.

The Delete button empties the playlist.

The Help displays a minimal on-line manual.

Play starts playing at the beginning of the play list, or at the selected track in the play list.

During play, Pause, Stop, Back, and Forward are active. Back and Forward go back or forward a
track at a time. The other buttons do the obvious thing.

The -t flag chooses a tiny font, useful for handhelds.

The -w flag creates the jukebox in a new window. Normally, the jukebox takes over the window in
which it is invoked.

The -s flag specifies the name under which the file descriptors of the playlist and database servers
are posted in /srv. This allows two or more play list servers to exist on one platform, e.g., when
there are several audio devices. The default value of the flag is $user for a playlist server at
/srv/playlists.$user and a database server at /srv/jukefs.$user.

Jukefs reads a set of maps describing the music data, builds an in-memory database, and pro-
vides lookup service to jukebox. The default map is /sys/lib/music/map. It consists of a
hierarchical set of objects. Each object has a type, a value, zero or more attribute-value pairs and
zero or more subobjects. An object consists of the type, followed by its contents between curly
brackets. Attribute value pairs consist of a single line containing an attribute name, an equals
sign, and a value. The value of an object is any text not containing curly brackets or equals signs.
Here is an example:

category {
    composer = mahler
}
Gustav Mahler
(1860 — 1911)

work {
    path {classic/mahler}
    class = symphonic
    orchestra = rfo
    conductor = Waart,-Edo-de

    Symphony Nº 5 in c (RFO, Vienna)
    performance{
        Radio Filharmonisch Orkest Holland
        Edo de Waart, conductor
        recorded: Musikverein, Vienna, May 6, 1996
    }
    command {number}
    track {
        Trauermarsch (In gemessenem Schritt. Streng. Wie ein Kondukt)
        time {13:55}
        file {034.pac}
    }
    track {
        Stürmisch bewegt, mit größter Vehemenz
        time {15:34}
        file {035.pac}
    }
    track {
        Scherzo (Kräftig, nicht zu schnell)
        time {18:54}
        file {036.pac}
    }
    track {
        Adagietto (Sehr Langsam)
        time {10:01}
        file {037.pac}
    }
    track {
        Rondo-Finale (Allegro)
        time {15:44}
        file {038.pac}
    }
}

This example shows a category object for the composer Gustav Mahler (the value consists of the two lines 'Gustav Mahler' and '1860 — 1911') with one subobject, a work object whose value is 'Symphony Nº 5 in c (RFO, Vienna)'. The work object contains six subobjects: one performance object and five track objects.

Category objects must contain exactly one attribute—value pair. The attribute names a subobject of the root under which this category object will be placed. Gustav Mahler, thus, will be placed in Root—composer. Work, Recording, Part, and Track, objects all describe named containers for subunits. A Lyrics, Performance, or Soloists object adds information to a Work, Recording, Part, or Track, object. It should only contain text. The same is true for a Time object; however, it should only be used adjacent to File objects and it should contain the running time of that file (this is for future use).

A File object specifies a file to be played. When the Select button is pressed, all file objects contained hierarchically in the selected object are added to the playlist.

There are a number of pseudo objects: Command may contain either sort or number. The sort command sorts the subobjects of the object it appears in by key or textual content. The number commands prepends numbers to the texts of its subobjects (e.g., for the parts in a symphony)

An Include object is replaced by the contents of the named file.

A Key object specifies a key for sorting subobjects.

Finally, a Path object specifies a path to be prepended to the files named in hierarchically contained File objects.
The attribute-value value pairs arrange for entries to be made of the current object in a Category object named by the attribute directly under the root.

The interface to the browsing database is through a file system implemented by jukefs. The file system synthesises a directory per object. Each directory contains a set of files describing the object’s attributes:

- **children**
  - contains a new-line separated list of subobject names. For each name, x the directory /mnt/ juke/x describes the subobject.

- **digest**
  - contains a one-line summary of the object

- **files**
  - is a new-line separated list of file objects contained in this object. Each line consists of object name and file name.

- **fulltext**
  - is the fulltextual value of the object.

- **key**
  - contains the key by which objects are sorted

- **miniparentage**
  - is a one-line summary of the objects and the path leading to it from the root. This is the line displayed in the playlist and bottom browse windows of games/jukebox.

- **parent**
  - is the object reference to the parent of this object.

- **parentage**
  - is a full description of the path leading to this object and the object itself. This is the string displayed in the top of the Browse and Playing windows of games/jukebox.

- **text**
  - is the text field of the object.

- **type**
  - is the type of the object

**FILES**

/sys/lib/music/map: Default map file

/mnt/juke: Default mount point for the music database.

**SOURCE**

/sys/src/games/music

**SEE ALSO**

playlistfs(7), audio(7)
NAME
map, mapdemo – draw maps on various projections

SYNOPSIS
map projection [ option ... ]
mapdemo

DESCRIPTION
Map prepares on the standard output a map suitable for display by any plotting filter described in plot(1). A menu of projections is produced in response to an unknown projection. Mapdemo is a short course in mapping.

The default data for map are world shorelines. Option –f accesses more detailed data classified by feature.

–f [ feature ... ]
Features are ranked 1 (default) to 4 from major to minor. Higher-numbered ranks include all lower-numbered ones. Features are
shore[1-4] seacoasts, lakes, and islands; option –f always shows shore1
ilake[1-2] intermittent lakes
river[1-4] rivers
iriver[1-3] intermittent rivers
canal[1-3] 3=irrigation canals

glacier
iceshelf[12]
reef
saltpan[12]
country[1-3] 2=disputed boundaries, 3=indefinite boundaries
state states and provinces (US and Canada only)

In other options coordinates are in degrees, with north latitude and west longitude counted as positive.

–l S N E W
Set the southern and northern latitude and the eastern and western longitude limits. Missing arguments are filled out from the list −90, 90, −180, 180, or lesser limits suitable to the projection at hand.

–k S N E W
Set the scale as if for a map with limits –l S N E W. Do not consider any –l or –w option in setting scale.

–o lat lon rot
Orient the map in a nonstandard position. Imagine a transparent gridded sphere around the globe. Turn the overlay about the North Pole so that the Prime Meridian (longitude 0) of the overlay coincides with meridian lon on the globe. Then tilt the North Pole of the overlay along its Prime Meridian to latitude lat on the globe. Finally again turn the overlay about its ‘North Pole’ so that its Prime Meridian coincides with the previous position of meridian rot. Project the map in the standard form appropriate to the overlay, but presenting information from the underlying globe. Missing arguments are filled out from the list 90, 0, 0. In the absence of –o, the orientation is 90, 0, m, where m is the middle of the longitude range.

–w S N E W
Window the map by the specified latitudes and longitudes in the tilted, rotated coordinate system. Missing arguments are filled out from the list −90, 90, −180, 180. (It is wise to give an encompassing –l option with –w. Otherwise for small windows computing time varies inversely with area!)

–d n For speed, plot only every nth point.
–r Reverse left and right (good for star charts and inside-out views).
-v Verso. Switch to a normally suppressed sheet of the map, such as the back side of the earth in orthographic projection.

-s1 Superpose; outputs for a -s1 map (no closing) and a -s2 map (no opening) may be concatenated.

-g dlat dlon res
    Grid spacings are dlat, dlon. Zero spacing means no grid. Missing dlat is taken to be zero. Missing dlon is taken the same as dlat. Grid lines are drawn to a resolution of res (2° or less by default). In the absence of -g, grid spacing is 10°.

-p lat lon extent
    Position the point lat, lon at the center of the plotting area. Scale the map so that the height (and width) of the nominal plotting area is extent times the size of one degree of latitude at the center. By default maps are scaled and positioned to fit within the plotting area. An extent overrides option -k.

-c x y rot
    After all other positioning and scaling operations have been performed, rotate the image rot degrees counterclockwise about the center and move the center to position x, y, where the nominal plotting area is \(-1 \leq x \leq 1, -1 \leq y \leq 1\). Missing arguments are taken to be 0. -x Allow the map to extend outside the nominal plotting area.

-m [file ...]
    Use map data from named files. If no files are named, omit map data. Names that do not exist as pathnames are looked up in a standard directory, which contains, in addition to the data for -f,

    world World Data Bank I (default)
    states US map from Census Bureau
    counties US map from Census Bureau

    The environment variables MAP and MAPDIR change the default map and default directory.

-b [lat0 lon0 lat1 lon1...]
    Suppress the drawing of the normal boundary (defined by options -l and -w). Coordinates, if present, define the vertices of a polygon to which the map is clipped. If only two vertices are given, they are taken to be the diagonal of a rectangle. To draw the polygon, give its vertices as a -u track.

-t file ...
    The files contain lists of points, given as latitude–longitude pairs in degrees. If the first file is named -, the standard input is taken instead. The points of each list are plotted as connected ‘tracks’.

    Points in a track file may be followed by label strings. A label breaks the track. A label may be prefixed by ",", ;, or ! and is terminated by a newline. An unprefixed string or a string prefixed with " is displayed at the designated point. The first word of a : or ! string names a special symbol (see option -y). An optional numerical second word is a scale factor for the size of the symbol, 1 by default. A : symbol is aligned with its top to the north; a ! symbol is aligned vertically on the page.

-u file ...
    Same as -t, except the tracks are unbroken lines. (-t tracks appear as dot–dashed lines if the plotting filter supports them.)

-y file
    The file contains plot(6)-style data for : or ! labels in -t or -u files. Each symbol is defined by a comment : name then a sequence of m and v commands. Coordinates (0,0) fall on the plotting point. Default scaling is as if the nominal plotting range were \(x a -1 -1 1 1; r a \) commands in file change the scaling.

Projections
    Equatorial projections centered on the Prime Meridian (longitude 0). Parallels are straight horizontal lines.
mercator
equally spaced straight meridians, conformal, straight compass courses
sinusoidal
equally spaced parallels, equal–area, same as bonne 0.
cylindrical
azimuthal
equally spaced straight meridians, equal–area, true scale on lat0
rectangular lat0
central projection on tangent cylinder
gastigomphal
equally spaced parallels, equally spaced straight meridians, true scale on lat0
mollweide
(parabolic) equal–area, hemisphere is a circle
gilbert() arc conformally mapped on hemisphere and viewed orthographically
gilbert
globe mapped conformally on hemisphere, viewed orthographically
Azimuthal projections centered on the North Pole. Parallels are concentric circles. Meridians are equally spaced radial lines.

azequidistant
equally spaced parallels, true distances from pole
azequalarea
equal–area
gnomonic
central projection on tangent plane, straight great circles
perspective dist
viewed along earth’s axis dist earth radii from center of earth
orthographic
viewed from infinity
stereographic
conformal, projected from opposite pole
laue radius = tan(2 × colatitude), used in X-ray crystallography
fisheye n
radius = log(colatitude/r): New Yorker map from viewing pedestal of radius r degrees
newyorker r
Polar conic projections symmetric about the Prime Meridian. Parallels are segments of concentric circles. Except in the Bonne projection, meridians are equally spaced radial lines orthogonal to the parallels.

conic lat0
central projection on cone tangent at lat0
simpleconic lat0 lat1
equally spaced parallels, true scale on lat0 and lat1
lambert lat0 lat1
conformal, true scale on lat0 and lat1
albers lat0 lat1
equal–area, true scale on lat0 and lat1
bonne lat0
equally spaced parallels, equal–area, parallel lat0 developed from tangent cone

Projections with bilateral symmetry about the Prime Meridian and the equator.
polyconic
parallels developed from tangent cones, equally spaced along Prime Meridian
aitoff
equal–area projection of globe onto 2–to–1 ellipse, based on azequalarea
lagrange
conformal, maps whole sphere into a circle
bicentric lon0
points plotted at true azimuth from two centers on the equator at longitudes ±lon0, great circles are straight lines (a stretched gnomonic)
elliptic lon0
points plotted at true distance from two centers on the equator at longitudes ±lon0
globular
hemisphere is circle, circular arc meridians equally spaced on equator, circular arc parallels equally spaced on 0– and 90-degree meridians
vandergrinten
sphere is circle, meridians as in globular, circular arc parallels resemble mercator

Doubly periodic conformal projections.
guyou
W and E hemispheres are square
square
world is square with Poles at diagonally opposite corners
tetra
map on tetrahedron with edge tangent to Prime Meridian at S Pole, unfolded into equilateral triangle
hex
world is hexagon centered on N Pole, N and S hemispheres are equilateral triangles

Miscellaneous projections.
harrison dist angle  oblique perspective from above the North Pole, dist earth radii from center of earth, looking along the Date Line angle degrees off vertical

trapezoidal lat0 lat1  
equally spaced parallels, straight meridians equally spaced along parallels, true scale at lat0 and lat1 on Prime Meridian

lune(lat,angle) conformal, polar cap above latitude lat maps to convex lune with given angle at 90°E and 90°W

Retroazimuthal projections. At every point the angle between vertical and a straight line to 'Mecca', latitude lat0 on the prime meridian, is the true bearing of Mecca.

mecca lat0  
equally spaced vertical meridians

homing lat0  
distances to Mecca are true

Maps based on the spheroid. Of geodetic quality, these projections do not make sense for tilted orientations. For descriptions, see corresponding maps above.

sp_mercator
sp_albers lat0 lat1

EXAMPLES

map perspective 1.025 –o 40.75 74
A view looking down on New York from 100 miles (0.025 of the 4000–mile earth radius) up. The job can be done faster by limiting the map so as not to ‘plot’ the invisible part of the world: map perspective 1.025 –o 40.75 74 –l 20 60 30 100. A circular border can be forced by adding option –w 77.33. (Latitude 77.33° falls just inside a polar cap of opening angle arccos(1/1.025) = 12.6804°.)

map mercator –o 49.25 –106 180
An ‘equatorial’ map of the earth centered on New York. The pole of the map is placed 90° away (49.25+49.25=90) on the other side of the earth. A 180° twist around the pole of the map arranges that the ‘Prime Meridian’ of the map runs from the pole of the map over the North Pole to New York instead of down the back side of the earth. The same effect can be had from map mercator –o 130.75 74

map albers 28 45 –l 20 50 60 130 –m states
A customary curved–latitude map of the United States.

map harrison 2 30 –l –90 90 120 240 –o 90 0 0
A fan view covering 60° on either side of the Date Line, as seen from one earth radius above the North Pole gazing at the earth’s limb, which is 30° off vertical. The –o option overrides the default –o 90 0 180, which would rotate the scene to behind the observer.

FILES
/lib/map/[1-4]?? World Data Bank II, for –f
/lib/map/* maps for –m
/lib/map/*.x map indexes
/bin/aux/mapd Map driver program

SOURCE
/sys/src/cmd/map

SEE ALSO
map(6), plot(1)

DIAGNOSTICS
‘Map seems to be empty’—a coarse survey found zero extent within the –l and –w bounds; for maps of limited extent the grid resolution, res, or the limits may have to be refined.

BUGS
Windows (option –w) cannot cross the Date Line. No borders appear along edges arising from visibility limits. Segments that cross a border are dropped, not clipped. Excessively large scale or –d setting may cause long line segments to be dropped. Map tries to draw grid lines dotted and –t tracks dot–dashed. As very few plotting filters properly support curved textured lines, these lines are likely to appear solid. The west–longitude–positive convention betrays Yankee chauvinism. Gilbert should be a map from sphere to sphere, independent of the mapping from sphere to plane.
NAME
playlistfs - playlist file system

SYNOPSIS
games/playlistfs [-s postname] [-m mountpoint] [-a]

DESCRIPTION
Playlistfs implements an audio player which plays files from a built-in play list. The player is
controlled through three files, usually mounted at /mnt. The files are /playctl for controlling play:
start, stop, pause, skip, etc.; /playvol for controlling the playout volume; and
/playlist for controlling the play list itself.

All three files can be written to control the player and read to obtain player status information.

When read, the files report the current status of the player, volume and playlist, respectively. End
of file is indicated by a read that returns zero bytes, as usual. However, in all three files, subse-
quent read operations will block until the status of the file changes and then report the changed
state. When the changed state has been read, another end-of-file indication is given, after which
another read can be issued to wait for state changes.

The /playctl file returns strings of the form ‘cmd n’ where cmd is one of stop, pause, or play
and n is an index (or offset) into the playlist; indices start at zero.

The commands that can be written to /playctl take the same form; however, the index is an
optional argument. If the index is omitted, the current value is used. The commands are play,
stop, pause, resume, and skip. Play starts playing at the index. Stop stops playing. If an index is
given, the current index is set to it and can be used in future commands. Pause and Resume inter-
rupt and continue play, respectively. The index argument is always ignored and the whole com-
mand is ignored if the state in which they occur does not make sense. Skip adds the argument to
the current index (adds one if no argument is given) and starts play at that index, stopping current
play, if necessary.

Reads of /playvol return strings of the form ‘volume n’, where n is a number or, if there is
more than one channel, a quoted set of numbers, between 0 (minimum) and 100 (maximum).
 Writes to /playvol take the same form.

The /playlist file is an append-only file which accepts lines with one or two fields per line
(parsed using tokenize). The first, compulsory, field is a file name, the optional second argu-
ment may contain a reference to, or a description of, the item, for instance in a graphical user
interface. /playlist is append-only, individual lines cannot be removed. However, the playlist
can be cleared by opening the file with the OTRUNC flag. A process that has /playlist open
while the file is truncated will receive an error on the next read with errstr set to reading past
eof. When this error occurs, clients can seek to the beginning of the file and reread its contents.

After starting up, Playlistfs puts itself in the background. When called with the -s flag, it
posts a mountable file descriptor in /srv/playlist.postname. The -m flag can be used to
specify a mount point other than /mnt.

The files to be played are recognized by one of four extensions, and an appropriate player is then
selected to play them. Files without a recognized extension are played by the pac player:

.mp3 /bin/games/mp3dec
.pac /bin/games/pac4dec
.pcm /bin/cp
.ogg /bin/games/vorbisdec

FILES
/srv/playlistfs.user default playlistfs mountable file descriptor used by juke(7).
/mnt/playctl: Control file
/mnt/playlist: Playlist file
/mnt/playvol: Volume control file

SOURCE
/sys/src/games/music/playlistfs
SEE ALSO

juke(7), audio(7)
NAME
  scat – sky catalogue and Digitized Sky Survey

SYNOPSIS
  scat

DESCRIPTION
  Scat looks up items in catalogues of objects outside the solar system and implements database-like manipulations on sets of such objects. It also provides an interface to astro(7) to plot the locations of solar system objects. Finally, it displays images from the Space Telescope Science Institute’s Digitized Sky Survey, keyed to the catalogues.

Items are read, one per line, from the standard input and looked up in the catalogs. Input is case-insensitive. The result of the lookup becomes the set of objects available to the database commands. After each lookup or command, if more than two objects are in the set, scat prints how many objects are in the set, otherwise it prints the objects’ descriptions or cross-index listings (suitable for input to scat). An item is in one of the following formats:

ngc1234
  Number 1234 in the New General Catalogue of Nonstellar Objects, NGC2000.0. The output identifies the type (Gx=galaxy, P1=planetary nebula, OC=open cluster, Gb= globular cluster, Nb= bright nebula, C+N=cluster associated with nebulosity, As=asterism, Kt=knot or nebuloser region in a galaxy, ***=triple star, D*=double star, ?= uncertain, -= nonexistent, PD=plate defect, and (blank)=unverified or unknown), its position in 2000.0 coordinates, its size in minutes of arc, a brief description, and popular names.

ic1234
  Like NGC references, but from the Index Catalog.

saol2345
  Number 12345 in the Smithsonian Astrophysical Star Catalogue. Output identifies the visual and photographic magnitudes, 2000.0 coordinates, proper motion, spectral type, multiplicity and variability class, and HD number.

m4
  Catalog number 4 in Messier’s catalog. The output is the NGC number.

abell1701
  Catalog number 1701 in the Abell and Zwicky catalog of clusters of galaxies. Output identifies the magnitude of the tenth brightest member of the cluster, radius of the cluster in degrees, its distance in megaparsecs, 2000.0 coordinates, galactic latitude and longitude, magnitude range of the cluster (the ‘distance group’), number of members (the ‘richness group’), population per square degree, and popular names.

planetarynebula
  The set of NGC objects of the specified type. The type may be a compact NGC code or a full name, as above, with no blank.

"α umi"
  Names are provided in double quotes. Known names are the Greek letter designations, proper names such as Betelgeuse, bright variable stars, and some proper names of stars, NGC objects, and Abell clusters. Greek letters may be spelled out, e.g. alpha. Constellation names must be the three-letter abbreviations. The output is the SAO number. For non-Greek names, catalog numbers and names are listed for all objects with names for which the given name is a prefix.

12h34m −16
  Coordinates in the sky are translated to the nearest ‘patch’, approximately one square degree of sky. The output is the coordinates identifying the patch, the constellations touching the patch, and the Abell, NGC, and SAO objects in the patch. The program prints sky positions in several formats corresponding to different precisions; any output format is understood as input.

umi
  All the patches in the named constellation.

mars
  The planets are identified by their names. The names shadow and comet refer to the Earth’s penumbra at lunar distance and the comet installed in the current astro(7). The
output is the planet's name, right ascension and declination, azimuth and altitude, and phase for the moon and sun, as shown by astro. The positions are current at the start of scat's execution; see the astro command in the next section for more information.

The commands are:

add item Add the named item to the set.

keep class ...
Flatten the set and cull it, keeping only the specified classes. The classes may be specific NGC types, all stars (sao), all NGC objects (ngc), all M objects (m), all Abell clusters (abell), or a specified brightness range. Brightness ranges are specified by a leading > or < followed by a magnitude. Remember that brighter objects have lesser magnitudes.

drop class ...
Complement to keep.

flat Some items such as patches represents sets of items. Flat flattens the set so scat holds all the information available for the objects in the set.

print Flatten the set, expand the area of the sky covered by the set to be n degrees wider, and collect all the objects in that area. If n is zero, expand collects all objects in the patches that cover the current set.

astro option Run astro(7) with the specified options (to which will be appended −p), to discover the positions of the planets. Astro's −d and −l options can be used to set the time and place; by default, it's right now at the coordinates in /lib/sky/here. Running astro does not change the positions of planets already in the display set, so astro may be run multiple times, executing e.g. add mars each time, to plot a series of planetary positions.

plot option Expand and plot the set in a new window on the screen. Symbols for NGC objects are as in Sky Atlas 2000.0, except that open clusters are shown as stippled disks rather than circles. Abell clusters are plotted as a triangle of ellipses. The planets are drawn as disks of representative color with the first letter of the name in the disk (lower case for inferior planets; upper case for superior); the sun, moon, and earth's shadow are unlabeled disks. Objects larger than a few pixels are plotted to scale; however, scat does not have the information necessary to show the correct orientation for galaxies. The option nongrid suppresses the lines of declination and right ascension. By default, scat labels NGC objects, Abell clusters, and bright stars; option nolabel suppresses these while all1label labels stars with their SAO number as well. The default size is 512×512; options dx n and dy n set the x and y extent. The option zenithup orients the map so it appears as it would in the sky at the time and location used by the astro command (q.v.). The output is designed to look best on an LCD display. CRTs have trouble with the thin, grey lines and dim stars. The option nogrey uses white instead of grey for these details, improving visibility at the cost of legibility when plotting on CRTs.

plate [[ra dec] rasize [decsize]] Display the section of the Digitized Sky Survey (plate scale approximately 1.7 arcseconds per pixel) centered on the given right ascension and declination or, if no position is specified, the current set of objects. The maximum area that will be displayed is one degree on a side. The horizontal and vertical sizes may be specified in the usual notation for angles. If the second size is omitted, a square region is displayed. If no size is specified, the size is sufficient to display the centers of all the objects in the current set. If a single object is in the set, the 500×500 pixel block from the survey containing the center of the object is displayed. The survey is stored in the CD-ROM juke box; run 9fs juke before running scat.

gamma value Set the gamma for converting plates to images. Default is −1.0. Negative values display white stars, positive black. The images look best on displays with depth 8 or greater. Scat does not change the hardware color map, which should be set externally to a grey scale; try the command getmap gamma (see getmap(9.1)) on an 8-bit color-mapped
EXAMPLES
Plot the Messier objects and naked-eye stars in Orion.
ori
keep m <6
plot nogrid

Draw a finder chart for Uranus:
uranus
expand 5
plot

Show a partial lunar eclipse:
astro -d
2000 07 16 12 45
moon
add shadow
expand 2
plot

Draw a map of the Pleiades.
"alcyone"
expand 1
plot

Show a pretty galaxy.
ngc1300
plate 10'

FILES
/lib/sky/*.scat

SOURCE
/sys/src/cmd/scat

SEE ALSO
astro(7)
/lib/sky/constelnames the three-letter abbreviations of the constellation names.

The data was provided by the Astronomical Data Center at the NASA Goddard Space Flight Center, except for NGC2000.0, which is Copyright © 1988, Sky Publishing Corporation, used (but not distributed) by permission. The Digitized Sky Survey, 102 CD-ROMs, is not distributed with the system.
NAME
   intro - introduction to system administration

DESCRIPTION
   This manual section describes commands for system administration as well as various utility programs necessary for the system but not routinely invoked by a user.
NAME
6in4 – configure and run automatic or manual 6to4 tunnel of IPv6 through IPv4

SYNOPSIS
ip/6in4 [−ag][−x netmtpt][local6[/mask][remote4 [remote6]]]

DESCRIPTION
6in4 sets up and maintains a tunnel of IPv6 traffic through an IPv4 connection.
Local6 and mask define the IPv6 address and subnet of the near end of the tunnel (mask defaults to /128 for a single-host tunnel). If local6 is missing or −, it defaults to
2002:aa bb cc dd
where aa, bb, cc and dd are the hexadecimal equivalents of the bytes a . b . c . d in this host’s primary IPv4 address.
Remote4 is the IPv4 address of the far end of the tunnel (must be given explicitly for a configured tunnel, or defaults to the anycast address 192.88.99.1 for 6to4).
Remote6 is the IPv6 address of the far end of the tunnel (used as the point–to–point destination for routing, and defaults to a link–local address constructed from remote4).
6in4 forks a pair of background processes to copy packets to and from the tunnel.
Options are:
−a permit any remote IPv4 address as the far end of a tunnel. This is likely to be useful for the server side of a tunnel.
−g use the tunnel as the default route for global IPv6 addresses
−x use the network mounted at netmtpt instead of /net.

EXAMPLES
If your primary IPv4 address is public, you can start a 6to4 tunnel simply with
ip/6in4 −g
Similarly, you can start a server for 6to4 tunnels with
ip/6in4 −ag
If you use a tunnel broker at address 5.6.7.8, configured to give you a /64 subnet with address 2001:1122:3344:5566::, you can start the tunnel with
ip/6in4 −g 2001:1122:3344:5566::/64 5.6.7.8

FILES
/net/ipmux access to IPv6–in–IPv4 packets
/net/ipifc packet interface to IPv6 network

SEE ALSO
bridge(3), ipmux in ip(3), linklocal in ipconfig(8)
/lib/rfc/rfc3056
/lib/rfc/rfc3068

BUGS
Needs a kernel with an ipmux driver.
The tunnel client filters addresses fairly conservatively in both directions. However it’s not watertight, and may be flakey in other ways so don’t put too much trust in it.
NAME
9load, 9pxeload, 9loadusb, 9loadask, Id – PC bootstrap program

SYNOPSIS
(Under MS–DOS)
[drive:] [path] Id [ 9load ]

DESCRIPTION
9load and Id are programs that reside in a FAT file system and bootstrap Plan 9. 9load loads the
kernel, but it cannot be run from DOS; use Id to bootstrap (by starting 9load) if DOS is running. 9load
is run automatically by the boot procedures described below; it cannot be run directly by
hand. 9pxeload is a version of 9load that can be booted using the PXE download (BOOTP/DHCP
followed by TFTP) found in some ethernet card BIOSes. 9loadusb is a version that will use only
the BIOS’s device drivers, and thus can load from USB devices. In contrast, 9load will not use BIOS
device drivers. 9loadask is a version that asks on the console (too early for serial ports, alas) if
you want to use BIOS drivers to boot. There are three bootstrap sequences:
- BIOS, MBR, disk partition PBS, 9load, kernel
- BIOS, floppy PBS, 9load, kernel
- BIOS, MBR, DOS, Id, 9load, kernel.

Details follow.

9load is a bootstrap program that loads and starts a program, typically the kernel, on a PC. It is
run by the PC partition boot sector program (PBS), which usually resides in the first sector of the
active partition. A copy of the Plan 9 PBS is kept in /386/pbs, but due to the “cylinder–head–sector”
(CHS) addressing mode of old BIOSes, it can only operate up to 8.5GB into the disk. Plan 9
partitions further into the disk can only be booted using /386/pbs1ba, and then only if the
machine’s BIOS supports linear block addressing (LBA) mode for disk transfers.

When booting from floppy or hard disk, the BIOS loads the first sector of the medium at location
0x7C00. In the case of a floppy, this is the PBS. In the case of a hard disk, it is the master boot
record (MBR). The MBR copies itself to address 0x600, finds the active partition and loads its PBS
at address 0x7C00. A copy of the Plan 9 MBR is kept in /386/mbz; some commercial MBRs cannot
read sectors past 2GB. The Plan 9 MBR can read sectors up to 8.5GB into the disk, and further
if the BIOS supports LBA. The single file /386/mbz detects whether the BIOS supports LBA and
acts appropriately, defaulting to CHS mode when LBA is not present. The PBSs cannot do this due
to code size considerations. The Plan 9 MBR is suitable for booting non–Plan 9 operating systems,
and (modulo the large disk constraints just described) non–Plan 9 MBRs are suitable for booting
Plan 9.

Thus the default sequence is: BIOS, MBR, PBS, 9load, kernel.

Because it contains many device drivers for different disks and networks, 9load is larger than 64K
and cannot be run as a DOS “.com” executable. A stripped–down version that knows about disks
but not networks, called Id (really 1d .cmd), fits in 64K and can be used under DOS to load and
start a program (default 9load) from the FAT16 partition. Its command line argument is of the
same format as the bootfile specifiers described below. This profusion of loaders is unfortunate,
but at least Id and 9load are compiled from the same source.

9load begins execution at virtual address 0x80010000 (64K) and loads the bootfile at the entry
address specified by the header, usually virtual 0x3F0100020. After loading, control is passed to
the entry location.

In summary, Plan 9 can be booted on a PC three different ways: either by booting MS–DOS and
using Id to start 9load in the appropriate directory, by booting directly from a Plan 9 boot floppy or
disk partition prepared using format to install the appropriate files and bootstrap sectors (see
prep(8)), or by using a PXE–capable BIOS to boot 9pxeload directly over the ethernet.

Bootfile
The bootfile, which may be compressed with gzip(1), can be specified to 9load as a bootfile= entry in
plan9.ini, or if booting from the ethernet, by a BOOTP server (see Kernel loading below). If the plan9.ini file contains multiple bootfile= entries, 9load will present a
numerical menu of the choices; type the corresponding number to select an entry.
The format of the `bootfile` name is `device ! file` or `device ! partition ! file`. If `! file` is omitted, the default for the particular `device` is used. Supported `devices` are:

- **fdn** An MS-DOS floppy disk. `N` specifies the floppy drive, either 0 or 1. The `bootfile` is the contents of the MS-DOS `file`. There is no default file. For compatibility with hard disks, a `partition` may be given, but only `dos` is recognized: `fd0 ! dos ! file`.

- **ethern** Ethernet. `N` specifies the Ethernet device number. If a `partition` is specified, it is taken to be the name of a host machine from which to load the kernel. `file` is determined by the `/lib/nxdb` (see `ndb(6)`) entry for this PC.

- **sdCn** Non–floppy disk. The device name format is described in `sd(3)`. A `partition` must be given and must name a partition containing a FAT file system. The name `dos` refers to the first DOS partition on a given device. It is common for Plan 9 partitions to contain a small FAT file system for configuration. By convention, this partition is called `9fat`. There is no default partition or pathname.

- **bios0** (Not in `pxeload`.) `9load` loads from a FAT file system on the first LBA device in the BIOS’s list of devices to try to boot from, using the BIOS INT 13 calls also used by `pslsba`. It does not understand any form of partition table; see the EXAMPLES in `prep(8)` for how to format such a device. This is mostly useful for booting from USB devices so far.

- **sdB0** (Not in `pxeload`) A special case of `sdCn` that uses `bios0` to read from a FAT file system. Partitions are understood.

### Kernel loading

When `9load` starts running at physical address 0x10000, it switches to 32–bit mode. It then double maps the first 16Mb of physical memory to virtual addresses 0 and 0x80000000. Physical memory from 0x3000000 upwords is used as data space.

`pxeload` differs slightly in operation from `9load`. It is initially loaded by the PXE BIOS at physical address 0x7C00. Only devices which can be automatically configured, e.g. most PCI ethernet adapters, will be recognised. If the file `/cfg/pxe/XXXXXXXXXXXX` can be located via a DHCP server, where `XXXXXXXXXXXX` is the MAC address of a recognised ethernet adapter, the contents are obtained and used as a `plan9.ini`.

Next, in order to find configuration information, `9load` searches all units on devices `fd` and `sdCn`, in that order, for a file called `plan9\plan9.ini` or `plan9.ini` (see `plan9.ini(8)`) on a partition named `dos` or `9fat`. If one is found, searching stops and the file is read into memory at physical address 0x1200 where it can be found later by any loaded `bootfile`. Some options in `plan9.ini` are used by `9load`:

- **console** Specifies the console device and baud rate if not a display.
- **baud**
- **ethern** Ethernet interfaces. These can be used to load the `bootfile` over a network. Probing for Ethernet interfaces is too prone to error.
- **bootfile=bootfile** Specifies the `bootfile`. This option is overridden by a command–line argument.
- **bootfile=auto** Default.
- **bootfile=local** Like `auto`, but do not attempt to load over the network.
- **bootfile=manual** After determining which devices are available for loading from, enter prompt mode.

When the search for `plan9.ini` is done, `9load` proceeds to determine which `bootfile` to load. If there was no `bootfile` option, `9load` chooses a default from the following prioritized device list:

- `fd` `sd` `ether`

`9load` then attempts to load the `bootfile` unless the `bootfile=manual` option was given, in which case prompt mode is entered immediately. If the default device is `fd`, `9load` will prompt the user for input before proceeding with the default `bootfile` load after 5 seconds; this prompt is omitted if a command–line argument or `bootfile` option was given.

`9load` prints the list of available `devices` and enters prompt mode on encountering any error or if directed to do so by a `bootfile=manual` option. In prompt mode, the user is required to type a `bootfile` in response to the `Boot from:` prompt.
Other facilities and caveats

9load parses the master boot record and Plan 9 partition tables (see prep(8)), leaving partitioning information appended to the in-memory contents of plan9.ini for the bootfile. This is used by sd(3) to initialize partitions so that fossil(4) or kfs(4) file systems can be mounted as the root file system. A more extensive partitioning is typically done by fdisk and prep as part of termrc or cpurc (see cpurc(8)).

A control–P character typed at any time on the console causes 9load to perform a hardware reset (Ctrl–Alt–Del can also be used on a PC keyboard).

When loaded from a PBS (rather than from ld.com), 9load must be contiguously allocated on the disk. See dossrv(4) for information on ensuring this.

FILES

[drive:][path]9load
[drive:][path]ld
FAT–filesystem:\plan9\plan9.ini
FAT–filesystem:\plan9.ini
/cfg/pxe directory of plan9.ini files on your TFTP server

SOURCE
/sys/src/boot/pc

SEE ALSO
booting(8), dhcpd(8), plan9.ini(8), prep(8)

BUGS

Much of the work done by 9load is duplicated by the loaded kernel.

If ld detects an installed MS–DOS Extended Memory Manager, it attempts to de–install it, but the technique used may not always work. It is safer not to install the Extended Memory Manager before running ld.

BIOS bugs force some limitations on reading via the BIOS. bios0 and sdB0 only work on the first LBA device in the BIOS’s list of boot devices.
NAME
9pcon - 9P to text translator

SYNOPSIS
aux/9pcon [-cn] [-m msize] service

DESCRIPTION
9pcon provides a textual interface to service, a conventional 9P server. By default, 9pcon interprets service as a file to be opened. The -c flag causes 9pcon to interpret service as a command to run which will carry out a (binary) 9P conversation over file descriptors 0 and 1. The -n flag causes 9pcon to interpret service as a network address to dial.

Once the connection is established, 9pcon prints R-messages as they arrive from the server, and sends T-messages as they are typed on standard input. There is no prompt. Lines beginning with # are ignored. The syntax for T-messages is one of:

  Tversion msize version
  Tauth afid uname aname
  Tatattach fid afid uname aname
  Twalk fid newfid wname...
  Topen fid mode
  Tcreate fid name perm mode
  Tread fid offset count
  Twrite fid offset data
  Tclunk fid
  Tremove fid
  Tstat fid
  Twstat fid name uid gid mode mtime length
  Tflush oldtag

See intro(5) for a description of the fields in each message. For the most part, the syntax mirrors the description of the messages in section 5. The exceptions are that the tags on the T-messages are added automatically; Twalk's wname count is inferred from the number of wnames given; and Twstat's dir is in expanded form rather than being an opaque byte sequence. Note that since commands are parsed with tokenize (see getfields(2)), it is easy to pass empty strings for absent name, uid, and gid fields. To ease specifying default integer fields, the Twstat message recognizes ~0 in the mode, mtime, and length arguments. For example,

  Twstat 101 ' ' ' ' sys ~0 ~0 ~0

sends a wstat message that attempts to change the group id associated with fid 101.

SOURCE
/sys/src/cmd/aux/9pcon.c

SEE ALSO
intro(5)

BUGS
There should be a flag to wait for responses, to facilitate scripting.
NAME
aan – always available network

SYNOPSIS
aan −c [−d][−m maxto ] dialstring
aan [−d][−m maxto ] netdir

DESCRIPTION
Aan tunnels traffic between a client and a server through a persistent network connection. If the connection breaks (voluntarily or due to networking problems), the aan client re-establishes the connection by redialing the server.

Aan uses a unique protocol to make sure no data is ever lost even when the connection breaks. After a reconnection, aan retransmits all unacknowledged data between client and server.

A connection can be broken voluntarily (e.g. by roaming over IP networks), or a connection can break when the IP service is unreliable. In either case, aan re-establishes the client’s connection automatically.

When the server part has not heard from the client in maxto seconds, the server part of aan exits. The default maxto is one day. The client side (option −c) calls the server by its dialstring, while the server side listens for connections in the already-announced network directory netdir.

Aan is usually run automatically through the −p option of import(4).

EXAMPLE
Assume the server part of aan is encapsulated in exportfs on the machine sob and started through aux/listen as follows:

```
netdir='{echo $3 | sed 's;/[0-9]+$/!*!0;'}
exec exportfs −a −A $netdir
```

Then machine astro6’s name space can be imported through aan using this command:

```
import −p astro6 / /mnt/term
```

FILES
/sys/log/aan Log file

SOURCE
/sys/src/cmd/aan.c

SEE ALSO
import(4), exportfs(4)
NAME
aliasmail - expand system wide mail aliases

SYNOPSIS
upas/aliasmail arg ...

DESCRIPTION
Aliasmail expands mail aliases, its arguments, according to alias files. Aliasmail is normally invoked by a rule in the upas rewrite file, rewrite(6).

If a line of an alias file begins with #include, the line is replaced by the contents of the file whose name follows. Other lines, beginning with # are ignored as comment.

Otherwise, lines begin with a name. The rest of a name line gives the expansion. The expansion may contain multiple addresses and may be continued to another line by appending a backslash. Items are separated by white space.

The alias files are searched in the order they are listed, one per line, in /mail/lib/namefiles. If the name is not found, the expansion is taken to be local! name. Under the -f option, alias files listed in /mail/lib/fromfiles are consulted instead, and the domain part only of the expansion is printed.

FILES
/mail/lib/namefiles names of system alias files

SOURCE
/sys/src/cmd/upas/alias

SEE ALSO
faces(1), filter(1), mail(1), marshal(1), mlmgr(1), nedmail(1), qer(8), rewrite(6), send(8), smtp(8), upasfs(4)
NAME

aoesrv – serve data via ATA-over-Ethernet (AoE)

SYNOPSIS

aoesrv [-i][-a shelf . slot][-c config][-e ether][-s size] file

DESCRIPTION

Aoesrv serves file via AoE (ATA-over-Ethernet) to the local Ethernet segment connected to the specified interfaces. File then appears on the network as an AoE target. Since AoE uses raw Ethernet frames (no IP), it is an unrouted protocol.

Options are:

- a Specify the shelf and slot (or major and minor) address of the target. Valid shelf numbers are between 0 and 65534. Valid slots are 0—255.
- c Set the AoE config string.
- e Listen to the network port ether. This option may be repeated to specify multiple ports.
- i Initialize the configuration header in file. All previous configuration information is lost. Without this option, configuration is read from file and command–like options override previous settings.
- s The exported target will be of size, rather than the available space in the target. A size may end in p, t, g, m, or k to specify a customary binary multiplier.

SEE ALSO

aoe(3), sdao(3)

BUGS

Security depends on control of a local Ethernet segment. It may be unwise to serve AoE on a segment bridged to a wireless network.
NAME
  apm - Advanced Power Management 1.2 BIOS interface

SYNOPSIS
  (in plan9.ini) apm0=
  bind -a '#P' /dev
  aux/apm [-d device] [-m mountpoint] [-s service]

DESCRIPTION
  Aux/apm presents at mountpoint (default /mnt/apm) an interface to the
  APM 1.2 BIOS (see apm(3)) device (the default is to try /dev/apm, followed by
  #P/apm). If a service is specified, the interface will be posted at /srv/service as well.

  The directory contains the following files.

  battery
    Contains one line for each battery in the system. Each line lists three fields: the status
    (a string, one of unknown, high, low, critical, or charging), the percent charge
    remaining, and an estimate of the amount of time left in seconds. If either or both of the
    last two are unknown, the corresponding field will be zero.

  ctl
    The ctl file is used to set power management modes for various parts of the system.
    Control messages are of the form "device verb," where device is one of system,
    display, storage, lpt, eia, network, and pcmcia, and verb is one of enable,
    disable, standby, and on. Enable and disable control whether power manage-
    ment is active for the device, while standby puts the device into standby mode and on
    brings it back to full power.

  event
    Reads from this file will block until an APM event has occurred. A large enough read is
    guaranteed to return an integral number of textual event descriptions, one per line.

SOURCE
  /sys/src/cmd/aux/apm.c
  /acme/bin/Battery

BUGS
  The verbs suspend and off should be supported but doing so requires nontrivial help from the
  kernel.
NAME
aquarela - CIFS server

SYNOPSIS
aquarela [ -np][ -d debug][ -u N][ -w workgroup]

DESCRIPTION
Aquarela provides CIFS (once known as SMB) access to Plan 9 file servers. It announces and subsequent-ly listens on tcp!*:1445 for connections to the file hierarchies called 'shares' by CIFS. Users are authenticated with their APOP secret (see auth(8)). Each session is managed by a process running as the authenticated user. Two persistent processes handle listening, session setup, and housekeeping.

An aquarela CIFS share corresponds to a directory under /n. A client request for a specific share, say, share, causes aquarela to attempt a 9fs (in srv(4)) connection to the file server share. If connection succeeds, a file hierarchy rooted at /n/share is presented to the client. The client request fails otherwise. Requests for the protocol equivalent of / are satisfied with a directory containing only the default share, local.

The options are:

- n Enable limited NETBIOS service. Aquarela will register with the 'master browser' for workgroup and listen on tcp!*:139 and udp!*:139[7-9] for NETBIOS name resolution and session requests. This works in tandem with a complete NETBIOS master name server, like that provided by Unix nmbd(8). NETBIOS is not required for CIFS file service.
- u N Send Unicode. For N, 1 enables, 0 disables Unicoing of file names and metadata. There is no code page support, so aquarela emits UTF if N is 0.
- w workgroup
  Set workgroup (or 'primary domain') of server. Default PLAN9.

EXAMPLE
To start CIFS service on system plan9:

% aquarela -u 1 &

To then make the dump filesystem available as drive Y: on a Windows machine:

C:\>net use Y: \plan9.example.com\dump

FILES
/n/local Default CIFS share
/sys/log/aquarela Log file

SOURCE
/sys/src/cmd/aquarela

SEE ALSO
auth(8), srv(4), utf(6)

DIAGNOSTICS
Log messages are appended to /sys/log/aquarela if it exists. The -p option prints them on standard output as well. The -d option adds verbose output about debug to the log messages, where debug is one of:

allcmds All CIFS requests and responses.
tids Connections and disconnections per-share.
sids Creation and deletion of search ids.
fids Creation and deletion of file ids.
rap2 RAP calls.
find Transaction2 find commands.
query Transaction2 query commands.
sharedfiles All files opened.
poolparanoia Draconian error checking in memory allocator.
sessions Connections and disconnections on server.
rep Regular expression conversions.
locks  Locking activity.

*any-smb-name*  Debug only SMB_ requests or transaction2 sub-requests (e.g., SMB_COM_SESSION_SETUP_ANDX or SMB_TRANS2_FIND_FIRST2) matching symbolic name *any-smb-name*.

0xnn  Debug only messages with hexadecimal id 0xnn.

**BUGS**

The first connection attempt to a share sometimes fails erroneously – try again. The share disk space reported by some clients is inaccurate. Some clients can’t rename directories. Write attempts without sufficient permissions fail (correctly), but appear on client to temporarily succeed.

This program should probably be named *cifsserver*. 
NAME
changeuser, convkeys, convkeys2, printnetkey, status, enable, disable, authsrv, guard.srv, debug, wrkey, login, newns, none, as – maintain or query authentication databases

SYNOPSIS
auth/changeuser [–np] user
auth/convkeys [–p] keyfile
auth/convkeys2 [–p] keyfile
auth/printnetkey user
auth/status user
auth/enable user
auth/disable user
auth/authsrv
auth/guard.srv
auth/debug
auth/wrkey
auth/login user
auth/newns [–ad] [–n namespace] command arg ...
auth/none [–n namespace] command arg ...
auth/as user command

DESCRIPTION
These administrative commands run only on the authentication server. Changeuser manipulates an authentication database file system served by keyfs(4) and used by file servers. There are two authentication databases, one holding information about Plan 9 accounts and one holding SecureNet keys. A user need not be installed in both databases but must be installed in the Plan 9 database to connect to a Plan 9 service.

Changeuser installs or changes user in an authentication database. It does not install a user on a Plan 9 file server; see fs(8) for that.

Option –p installs user in the Plan 9 database. Changeuser asks twice for a password for the new user. If the responses do not match or the password is too easy to guess the user is not installed. Changeuser also asks for an APOP secret. This secret is used in the APOP (RFC1939), CRAM (RFC2195), and Microsoft challenge/response protocols used for POP3, IMAP, and VPN access.

Option –n installs user in the SecureNet database and prints out a key for the SecureNet box. The key is chosen by changeuser.

If neither option –p or option –n is given, changeuser installs the user in the Plan 9 database.

Changeuser prompts for biographical information such as email address, user name, sponsor and department number and appends it to the file /adm/netkeys.who or /adm/keys.who.

Convkeys re-encrypts the key file keyfile. Re-encryption is performed in place. Without the –p option convkeys uses the key stored in NVRAM to decrypt the file, and encrypts it using the new key. By default, convkeys prompts twice for the new password. The –p forces convkeys to also prompt for the old password. The format of keyfile is described in keyfs(4).

The format of the key file changed between Release 2 and 3 of Plan 9. Convkeys2 is like convkeys. However, in addition to rekeying, it converts from the previous format to the Release 3 format.

Printnetkey displays the network key as it should be entered into the hand-held SecureNet box.

Status is a shell script that prints out everything known about a user and the user’s key status.

Enable/disable are shell scripts that enable/disable both the Plan 9 and Netkey keys for individual users.
Authsrv is the program, run only on the authentication server, that handles ticket requests on TCP port 567. It is started by an incoming call to the server requesting a conversation ticket; its standard input and output are the network connection. Authsrv executes the authentication server’s end of the appropriate protocol as described in authsrv(6).

Guard.srv is similar. It is called whenever a foreign (e.g. Unix) system wants to do a SecureNet challenge/response authentication.

**Anywhere commands**
The remaining commands need not be run on an authentication server.

**Debug** attempts to authenticate using each p9sk1 key found in factotum and prints progress reports.

**Wrkey** prompts for a machine key, host owner, and host domain and stores them in local non-volatile RAM.

**Login** allows a user to change his authenticated id to user. Login sets up a new namespace from /lib/namespace, starts a factotum(4) under the new id and execs rc(1) under the new id.

**Newns** sets up a new namespace from namespace (default /lib/namespace) and execs its arguments. If there are no arguments, it execs /bin/rc. Under –a, newns adds to the current namespace instead of constructing a new one. The –d option enables debugging output.

**None** sets up a new namespace from namespace (default /lib/namespace) as the user none and execs its arguments under the new id. If there are no arguments, it execs /bin/rc. It’s an easy way to run a command as none.

As executes command as user. Command is a single argument to rc, containing an arbitrary rc command. This only works for the hostowner and only if #!/caphash still exists.

**FILES**
/lib/ndb/auth Speaks for relationships and mappings for RADIUS server id’s.
/adm/keys.who List of users in the Plan 9 database.
/adm/netkeys.who List of users in the SecureNet database.
/sys/lib/htppasswords List of realms and passwords for HTTP access.

**SOURCE**
/sys/src/cmd/auth

**SEE ALSO**
passwd(1), readnram in authsrv(2), keyfs(4), seurenet(8)

**BUGS**
Only CPU kernels permit changing userid.
NAME
backup, tobackup, dumparenas, restore - backup venti arenas to blu-ray discs or restore from them

SYNOPSIS
backup [-n] [-d dev] [-s set]
tobackup [-set]
dumparenas dev arena ...
restore arena-# [dev]

DESCRIPTION
These programs reside in /sys/lib/backup and provide a means to backup venti(8) storage to Blu-ray (or other large optical) discs, while keeping track of which arenas have been written to which discs. Multiple backup sets are supported, as is (re)loading a venti store from a backup thus made.

The first time that backup is run, it will dump all sealed venti arenas. Thereafter, it will append only those sealed arenas not already written to a disc within the given backup set. The -s option uses a backup set other than the default. The -d option uses a disc burner other than the default /dev/sdD0. The -n option goes through the motions but does not burn any tracks on the Blu-ray disc. As a side-effect, backup run on the default dump set, set1, will also print the last few fossil dump scores.

Tobackup prints the names of all the sealed arenas not yet backed up to a disc in the current set.

Dumparenas copies the named arenas, one per track, to the device dev, which is first mounted via cdfs(4). Venti/rdarena is used to extract each arena.

Tobackup and dumparenas are invoked internally by backup.

Restore copies each data track (assumed to be a saved arena) on dev (by default, /dev/sdC0) into its appropriate place in the venti arenas partition (locally, /dev/sde0/arenas), thus adding the arena to the current venti store. The arena size of the arena partition must match the size of the arenas on optical disc (except for ~60K of trailing debris on the optical disc arenas).

Arena-# must be the number (starting from zero) of the first arena slot in the arenas partition that you wish to restore into from the current optical disc (not necessarily that of the first arena on the disc). Restore will prompt for confirmation that the first arena is the correct one, after printing a summary of its arena header. Typing y will proceed normally, n will abort all processing, and skip will cause restore to proceed to the next track and ask for confirmation of it.

The arenas partition must be formatted (see fmtarenas in venti–fmt(8)) before restoring into it. When all the arenas have been restored, it will be necessary to build a new venti index, the usual steps being to run checkarenas, fmtisect, fmtbloom, fmtindex, and buildindex -b, all from venti–fmt(8).

FILES
/sys/lib/backup backup scripts and records
set1 subdirectory containing records for default backup set
/sys/log source of dump scores

SOURCE
/sys/lib/backup

SEE ALSO
venti(1), cdfs(4), venti(8), venti–fmt(8), venti–backup(8)
Venti Backup on Blu-Ray Discs

BUGS
Assumes a single arenas partition named arena0. Assumes that the file server’s arenas are accessible on it as /dev/fs/arena0.
NAME
bandt2 - Viaduct bridge-and-tunnel client and server

SYNOPSIS
aux/bandt2 [-b bridge] [-c brickdb] [-h host] ... [-H hash] ... [-s secret] [-u user] [-x mtpt] [-Dr]

DESCRIPTION
Bandt2 establishes a compressed-then-encrypted tunnel over sdp(3) on UDP port 7331, bridging all traffic from a user's home systems to the Murray Hill Research network. Bandt2 is started in its client mode inside a Viaduct, which runs Plan 9 on a Power PC, and in its server mode on the Plan 9 systems vespine and vespine2, which terminate the Viaduct traffic and bridge it. Should the tunnel connection break, the bandt2 client will try repeatedly to re-establish it, when invoked with -r, which the Viaducts do.

Authentication is via 96-bit HMAC MD5 hash, encryption is 128-bit RC4, and compression uses the locally-developed thwack algorithm.

Other options are:
b use bridge rather than #B0/bridge0.
c use brickdb rather than /adm/brickdb.
D act as the bandt2 server (daemon) instead of the client.
h adds host to the set of hosts to call in round-robin fashion.
H adds hash to the set of host hashes. These hashes correspond to the hosts with the same ordinal values (i.e., the invoker must supply -h and -H in the same order).
s use secret rather than /adm/keydb/my.sec.
u use user instead of the system name returned by sysname in getuser(2).
x use the network mounted at mtpt instead of /net.

EXAMPLES
Start a server.
   aux/bandt2 -D
Start a client on a system with two Ethernet interfaces.
   echo -n bind ether '{cat /net.alt/ether1/addr} 0 /net.alt/ether1 \
   >/net/bridge0/ctl
   bind -a '#D' /net
   bind -a '#E' /net
   aux/bandt2 -r -h 135.104.9.65 -H some-hash -h 135.104.9.73 \
   -H 135.104.9.85 -r

FILES
/adm/brickdb database of which Viaducts are active and who owns them.
/adm/keydb/my.sec Viaduct's secret

SOURCE
/sys/src/cmd/aux/bandt2

SEE ALSO
bridge(3), sdp(3)
/sys/src/cmd/aux/bandt2/protocol.*

BUGS
Appears not to bridge IPv6, only IPv4 and ARP, due to limitations in bridge(3).
NAME
   boot - connect to the root file server

SYNOPSIS
   /boot/boot [-fkm] [-u username] [ method ! fs-addr ] [ args ]

DESCRIPTION
   Boot is the first program run after a kernel has been loaded. It connects
to the file server that will serve the root, performs any authentication
needed to connect to that server, and exec(2)'s the init(8) program. It is started
by the kernel, never run directly by the user. See booting(8) for information
about the process of loading the kernel (and boot) into memory.

   Once loaded, the kernel initializes its data structures and devices. It sets
the two environment variables /env/cputype and /env/terminal to describe the
processor. It then binds a place-holder file server, root(3), onto / and crafts an
initial process whose sole function is to exec(2) /boot/boot, a binary which is
compiled into root(3).

   The command line passed depends on the information passed from boot ROM
to kernel. Machines that boot directly from ROM (that is, most machines other
than PCs) pass the boot line given to the ROM directly to boot.

   On the PC, each line in the DOS file plan9.ini of the form name=value is
passed to the boot program as an environment variable with the same name and
value. The command line is

   /386/9dos method!server

   (The first argument is ignored by boot.) Boot must determine the file server
to use and a method with which to connect to it. Typically this will name a file
server on the network, or state that the root file system is on local disk and
name the partition. The complete list of methods is given below.

   Boot must also set a user name to be used as the owner of devices and all
console processes and an encryption key to be used when challenged. Boot will
prompt for these.

   Method and address are prompted for first. The prompt lists all valid
methods, with the default in brackets, for example:

   root is from (tcp, local!#S/sdC0/fs)[tcp]:

   A newline picks the default. Other possible responses are method or method!
address. To aid in automatic reboot, the default is automatically taken on CPU
servers if nothing is typed within 15 seconds.

   The other interactions depend on whether the system is a terminal or a
CPU server.

Terminal
   The terminal must have a username to set. If none is specified with the
-u option, boot will prompt for one on the console:

   user:

   The user will also be prompted for a password to be used as an encryption
key on each attach(5):

   password:

   With most methods boot can now connect to the file server. However, with
the serial line methods 9600 and 19200, the actual mechanics of setting up the
complete connection are too varied to put into the boot program. Instead boot
lets the user set up the connection. It prints a prompt on the
console and then simulates a dumb terminal between the user and the serial
line:

   Connect to file system now, type ctrl-d when done.
   (Use the view or down arrow key to send a break)

   The user can now type at the modem to dial the number. What is typed
depends on the modem and is beyond this discussion.

   When the user types a control-D, boot stops simulating a terminal and
starts the file system protocol over the serial line.

   Once connected, boot mounts the root file system before / and makes the
connection available as #s/boot for subsequent processes to mount (see bind(2)).
Boot completes by exec(2)'ing
Booting Methods

The methods available to any system depend on what was compiled into the kernel. The complete list of booting methods is listed below.

tcp   connect via Ethernet using the TCP protocol. The args are passed to ipconfig(8) when configuring the IP stack. The plan9.ini(8) variables fs and auth override the file server and authentication server IP addresses obtained (if any) from DHCP during ipconfig(8).

local connect to the local file system. The first argument is a disk holding a file system. Boot inspects the disk. If the disk is a fossil(4) file system, it invokes /boot/fo to serve it. If the venti environment variable (really, plan9.ini(8) variable) is set, boot first arranges for fossil to be able to contact the named venti(8) server. The variable’s value can take the following forms:

/dev/sdC0/arenas
the file should be a venti partition with a configuration stored on it using venti/conf (see venti-fmt(8)). Boot will start a loopback IP interface on 127.0.0.1 and start venti announcing on tcp!127.1!17034 for venti service and tcp!127.1!8000 for web service, using the configuration stored in that partition.

/dev/sdC0/arenas tcp!*!17034
same as the last but specify an alternate venti service address. In this example, using * will announce on all available IP interfaces (even ones configured later) rather than just the loopback device. The loopback interface is still configured.

/dev/sdC0/arenas tcp!*!17034 tcp!*!80
same as the last but specify alternate venti service and web addresses. The loopback interface is still configured.

tcp!135.104.9.2!17034 [args]
the network address of a venti server running on a separate machine. Boot will configure the IP stack by passing args, if any, to ipconfig(8).

If the disk is not a fossil(4) partition, boot invokes /boot/kfs. A variety of programs, like 9660srv and dosrv(4) masquerade as kfs to allow booting from alternate media formats, so as long as the disk is not a fossil disk, no check is made that the disk is in fact a kfs disk. The args are passed to kfs(4).

For the tcp method, the address must be a numeric IP address. If no address is specified, a file server address will be found from another system on the network using the BOOTP protocol and the Plan 9 vendor-specific fields.
EXAMPLES

On PCs, the default arguments to boot are constructed using the bootargs variable in plan9.ini.

Start kfs(4) with extra disk buffers:

    bootargs=local!#S/sdC0/fs −B 4096

Use an IP stack on an alternate ethernet interface with a static address and fixed file server and authentication server addresses.

    fs=192.168.0.2
    auth=192.168.0.3
    bootargs=tcp −g 192.168.0.1 ether /net/ether1 \ 192.168.0.50 255.255.255.0

(The bootargs line is split only for presentation; it is one line in the file.)

FILES

    #s/boot
    #/boot/boot

SOURCE

    /sys/src/9/boot

SEE ALSO

    root(3), dhcpd(8), init(8)

BUGS

The use of bootargs in general is odd. The configuration specification for fossil and venti servers is particularly odd, but it does cover the common cases well.
NAME
booting - bootstrapping procedures

SYNOPSIS
none

DESCRIPTION
This manual page collects the incantations required to bootstrap Plan 9 machines. Some of the
information here is specific to the installation at Bell Labs; some is generic.

If a CPU server is up, BOOTP/DHCP and TFTP will run from there; if not, the necessary files and ser-
VICES must be available on a separate machine, such as a Unix system, to use these protocols for
bootstrapping.

Be sure to read boot(8) to understand what happens after the kernel is loaded.

Terminals
To bootstrap a diskless terminal or a CPU server, a file server must be running. PCs can boot from
a floppy disk or any FAT16 partition. On all the terminals, typing two control-T’s followed by a
lower-case z reboots the machine; other methods of rebooting are mentioned for some machines.

PCs
To boot a PC, it is necessary to get /386/9load or /386/9pxelload loaded into memory.
There are many ways to do this. A Plan 9 boot floppy prepared by format (see prep(8)) will load
9load when the PC is reset or powered on. Other methods are described in 9load(8). 9load
then locates and loads a Plan 9 kernel, using configuration information from the file plan9.ini
stored in the 9fat configuration partition or on a DOS file system. See 9load(8) for details.

Once the kernel is booted, it behaves like the others. See boot(8) for details.

Alpha PCs
Alpha PCs must be booted via TFTP using the SRM console. If the system has ARC firmware
instead, SRM may be downloaded from

http://www.compaq.com/

You must configure the SRM firmware to load the file /alpha/bootalphapc. The following
commands may be used (replace ewa0 with the name of your ethernet device, if different):

set boot_reset ON
set boot_file /alpha/bootalphapc
set bootdef_dev ewa0
set ewa0_inet_init bootp
set ewa0_protocols BOOTP

This secondary bootstrap program will first load the file /alpha/conf/<IP-address> (substitut-
ing the IP address of the system as obtained via boot). This file is expected to be in plan9.ini(8)
format (the file /alpha/conf/10.0.0.2 may be used as a template). It then loads the kernel
via tftp, using the value of bootfile to tell it which file to load; this should be /alpha/9apc
for terminals.

CPU Servers
The Plan 9 CPU servers are multi-user, so they do not request a user name when booting. On the
CPU servers, typing a control-P on the console reboots the machine.

PC CPU Server
Proceed as for the PC terminal, but load /386/9pccpu or /386/9pccpudisk.

Alpha PC CPU Server
Proceed as for the Alpha PC terminal, but use /alpha/9apccpu as the value of bootfile.

SGI Challenge multiprocessor CPU Server
The Challenge ROM monitor can boot from the Ethernet. To boot from the Ethernet, type

bootp()/mips/9ch

or use the ROM command setenv to set the variable bootfile to that same string and type
boot. To load a different file, tell bootp which file to load, and to force the download to come
from a particular system, bootp()system:file. Any arguments after bootp()file are
passed to /boot. If you are running a Plan 9 BOOTP server (see *dhcpd*(8)), the file name can be omitted and the file specified by the bootf parameter for the machine in /lib/ndb will be downloaded by default.

Once the kernel is loaded, it prompts for the Ethernet protocol to use to reach the root file server; request the default.

**ARM CPU Servers**

All ARM systems are started by U-boot using similar commands. The kernels (and thus ndb bootf parameters) are /arm/9gd for the Marvell PXA168-based Guruplug Display, /arm/9plug for all Marvell Kirkwood plugs (Sheevaplug, Guruplug, Openrd, etc.), and /arm/9beagle for T1 OMAP3 boards (IGEPv2 from ISEE, Gumstix Overo). In the following, replace MAC with your board’s MAC address without colons, in lower case (the format of the ether ndb attribute).

First, establish a /cfg/pxe (plan9.ini) file for the new CPU server. For Kirkwood plugs,

```
cd /cfg/pxe; cp example-kw MAC
```

and edit /cfg/pxe/MAC to taste. For PXA plugs, replace kw with pxa; for OMAP boards, replace kw with omap and be sure to edit the line for ether0 to set

```
ea=MAC
```

Second, configure U-boot to load the appropriate kernel and /cfg/pxe file at suitable addresses and start the kernel. For Sheevaplugs and Openrd boards, type this at U-boot once:

```
setenv bootdelay 2
# type the next two lines as one
setenv bootcmd 'boot; bootp; tftp 0x1000 /cfg/pxe/MAC; bootp;
tftp 0x800000; go 0x800000'
saveenv
```

For Guruplug Displays, do the same but type this after setenv bootcmd instead:

```
'dhcp; tftpboot; tftpboot 0x1000 /cfg/pxe/MAC; bootz 0x500000'
```

For Kirkwood Guruplug, type this after setenv bootcmd:

```
dhcp 0x800000; tftp 0x1000 /cfg/pxe/MAC; go 0x800000
```

For IGEPv2 boards, type this after setenv bootcmd:

```
tftp 0x80300000 /cfg/pxe/MAC; dhcp 0x80310000; go 0x80310000
```

For Gumstix Overo boards, type this after setenv bootcmd:

```
'bootp 0x80310000; bootp 0x80300000 /cfg/pxe/MAC; go 0x80310000'
```

Thereafter, the boards will automatically boot via BOOTP and TFTP when reset.

**SEE ALSO**

*ndb*(6), *9load*(8), *boot*(8), *init*(8), *plan9.ini*(8)

**SOURCE**

Sources for the various boot programs are under /sys/src/boot.
NAME

   cec - Coraid Ethernet Console

SYNOPSIS

   cec [-d] [-S srv][-c esc][-e ea][-h host][-s shelf][-i interface]

DESCRIPTION

   Cec uses raw Ethernet packets to connect to a CEC server for console access. All clients share the same session. Coraid appliances and Coraid Plan 9 kernels can currently be CEC servers.

   Cec starts by probing the specified network interface for available CEC servers. The default is /net/ether0. Only one cec process may be run per Ethernet interface. If the server is specified with the -b, -h, or -s options, communication will proceed immediately upon discovery of the first CEC server with the specified address. Otherwise, a selection prompt will be displayed showing the discovered CEC servers available for communication. Unless the -p option is specified, cec exits if no matching servers are found. The selection prompt accepts

   number  Connect to server number (from the first column),
   p       Probe the interface again, and
   q       Quit.

   Note the selection number is not the shelf address but the cec-generated sequence number printed in the leftmost column.

   Once connected to a CEC server, typing the escape character will drop the user into an escape prompt where the user may type q to quit the connection, i to send the escape character across the connection, or . to continue the connection.

Options

   -c   Set the escape character to control-esc. The default setting is control-

   -d   Print debugging information.

   -e   Connect to the server with Ethernet address ea; implies -p.

   -h   Connect to the server host. Note that this name might not be the same as the contents of /dev/sysname on the target system.

   -p   Persist: continue trying to connect even if there are no matching servers. This is useful when connecting to a CPU server before it boots.

   -s   Connect to the server at address shelf.

   -S   Post the CEC connection as /srv/srv to allow sharing.

   If the -e, -s, or -h options are given, cec will exit upon closing the connection. Otherwise, cec will return to the selection prompt upon connection close.

EXAMPLES

   ; cec '#11/ether1'
      0 1 003048679b89
      [#qp]: 0

      SR shelf 1>
      SR shelf 1> >>> q
      0 1 003048679b89
      [#qp]: q
      ;

SOURCE

   /sys/src/cmd/cec

BUGS

   The CEC protocol should be integrated with the console server. The arbitration between the keyboard and network is suboptimal.

   Early boot information and very late crash information from servers may be lost due to timing quirks.
NAME
cpurc, cpurc.local, termrc, termrc.local - boot scripts

SYNOPSIS
cpurc
cpurc.local
termrc
termrc.local

DESCRIPTION
After the kernel boots, it execs /boot (see root(3)), which in turn execs /$cputype/init. Init(8) sets the $service environment variable to cpu or terminal, and then invokes the appropriate rc script to bring the system up.

Based on the values of $sysname and $terminal these scripts start appropriate network processes and administrative daemons and enable swapping. Cpurc sets /env/boottime to the time cpurc was executed and /env/NPROC to a value suitable for parallel compilation in mk(1).

If an executable file /bin/termrc.local exists, termrc will execute it. If an executable file /cfg/$sysname/termrc exists for the machine named $sysname, termrc will execute it next. Automatic initialization of the mouse and VGA on a PC is suppressed, if the user is none. These files should be edited by local installations to reflect the configuration of their systems.

On CPU servers, read cpurc for termrc in the previous paragraph.

FILES
/cfg/$sysname/cpurc  machine-specific boot script for cpurc
/cfg/$sysname/termrc  machine-specific boot script for termrc

SOURCE
/rc/bin/*rc
/rc/bin/*rc.local
/cfg/$sysname/*rc

SEE ALSO
   srv(4), namespace(6), init(8), listen(8)
NAME
cron - clock daemon

SYNOPSIS
auth/cron [-c]

DESCRIPTION
Cron executes commands at specified dates and times according to instructions in the files
/cron/user/cron. It runs only on an authentication server. Option -c causes cron to create
/cron/user and /cron/user/cron for the current user; it can be run from any Plan 9
machine.

Blank lines and lines beginning with # in these files are ignored. Entries are lines with fields

minute hour day month weekday host command

Command is a string, which may contain spaces, that is passed to an rc(1) running on host for
execution. The first five fields are integer patterns for

minute 0–59
hour 0–23
day of month 1–31
month of year 1–12
day of week 0–6; 0=Sunday

The syntax for these patterns is


time : '*'
| range
range : number
| number '-' number
| range ',' range

Each number must be in the appropriate range. Hyphens specify inclusive ranges of valid times;
commas specify lists of valid time ranges.

To run the job, cron calls host and authenticates remote execution, equivalent to running rx host
cmand (see con(1)). The user's profile is run with $service set to rx. If host is set to
local, cron will run the command as user on the local machine without using rx.

Cron is not a reliable service. It skips commands if it cannot reach host within two minutes, or if
the cron daemon is not running at the appropriate time.

EXAMPLES
Here is the job that mails system news.

% cat /cron/upas/cron
# send system news
15 8–17,21 * * * helix /mail/lib/mailnews
%

FILES
/cron/lock  lock file to prevent multiple crons running

SOURCE
/sys/src/cmd/auth/cron.c

SEE ALSO
con(1), rc(1)
NAME
dhcpd, dhcpleases, rarpd, tftp – Internet booting

SYNOPSIS
    ip/dhcpd [-dmnprsSZ] [-f ndb-file] [-M secs] [-x netmtpt] [-Z secs] [ address n ] ...
    ip/dhcpleases
    ip/rarpd [-d] [-e etherdev] [-x netmtpt]
    ip/tftp [-dr] [-h homedir] [-x netmtpt]

DESCRIPTION
These programs support booting over the Internet. They should all be run on the same server to
allow other systems to be booted. Dhcpd and tftp are used to boot everything; rarpd is an extra
piece just for Suns.

Dhcpd runs the BOOTP and DHCP protocols. Clients use these protocols to obtain configuration
information. This information comes from attribute/value pairs in the network database (see
ndb(6) and ndb(8)). DHCP requests are honored both for static addresses found in the NDB and for
dynamic addresses listed in the command line. DHCP requests are honored if either:
- there exists an NDB entry containing both the ethernet address of the requester and an IP
address on the originating network or subnetwork.
- a free dynamic address exists on the originating network or subnetwork.

A BOOTP request is honored if all of the following are true:
- there exists an NDB entry containing both the ethernet address of the requester and an IP
address on the originating network or subnetwork.
- the entry contains a bootf= attribute
- the file in the bootf= attribute is readable.

Dynamic addresses are specified on the command line as a list of addresses and number pairs. For example,
    ip/dhcpd 10.1.1.12 10 10.2.1.70 12
directs dhcpd to return dynamic addresses 10.1.1.12 through 10.1.1.21 inclusive and 10.2.1.70
through 10.2.1.81 inclusive.

Dhcpd maintains a record of all dynamic addresses in the directory /lib/ndb/dhcp, one file
per address. If multiple servers have access to this common directory, they will correctly coordinate
their actions.

Attributes come from either the NDB entry for the system, the entry for its subnet, or the entry for
its network. The system entry has precedence, then the subnet, then the network. The NDB
attributes used are:
    ip the IP address
    ipmask the IP mask
    ipgw the default IP gateway
    dom the domain name of the system
    fs the default Plan 9 name server
    auth the default Plan 9 authentication server
    dns a domain name server
    ntp a network time protocol server
    time a time server
    wins a NETBIOS name server
    www a World Wide Web proxy
    pop3 a POP3 mail server
    smtp an SMTP mail server
    bootf the default boot file; see ndb(6)

Dhcpd will answer BOOTP requests only if it has been specifically targeted or if it has read access to
the boot file for the requester. That means that the requester must specify a boot file in the
request or one has to exist in NDB for dhcpd to answer. Dhcpd will answer all DHCP requests for
which it can associate an IP address with the requester. The options are:
d  Print debugging to standard output.

f  Specify a file other than /lib/ndb/local as the network database.

m  Mute: don’t reply to requests, just log them and what dhcpd would have done.

M  Use secs as the minimum lease time for dynamic addresses.

n  Don’t answer BOOTP requests.

p  Answer DHCP requests from PPTP clients only.

r  Mute static addresses: don’t reply to requests for static addresses, just log them and what dhcpd would have done.

s  Sleep 2 seconds before answering requests for static addresses. This is used to make a server be a backup only.

S  Sleep 2 seconds before answering requests for dynamic addresses.

x  The IP stack to use is mounted at netmtpt. The default is /net.

Z  Use secs as the minimum lease time for static addresses.

Dhcpdleases prints out the currently valid DHCP leases found in the /lib/ndb/dhcp directory.

Rarpd performs the Reverse Address Resolution Protocol, translating Ethernet addresses into IP addresses. The options are:

d  Print debugging to standard output.

e  Use the Ethernet mounted at /net/etherdev.

x  The IP stack to use is mounted at netmtpt. The default is /net.

Tftpd transfers files to systems that are booting. It runs as user none and can only access files with global read permission. The options are:

d  Print debugging to standard output.

x  The IP stack to use is mounted at netmtpt. The default is /net.

h  Change directory to homedir. The default is /lib/tftpd. All requests for files with non-rooted file names are served starting at this directory with the exception of files of the form xxxxxxx.x.SUNyy. These are Sparc kernel boot files where xxxxxxxx is the hex IP address of the machine requesting the kernel and yy is an architecture identifier. Tftpd looks up the file in the network database using ipinfo (see ndb(2)) and responds with the boot file specified for that particular machine. If no boot file is specified, the transfer fails. Tftpd supports only octet mode.

r  Restricts access to only those files rooted in the homedir.

FILES
   /lib/ndb/dhcp directory of dynamic address files

SOURCE
   /sys/src/cmd/ip

SEE ALSO
   ndb(6), 9load(8), booting(8)
NAME
diskparts, dmaon - prepare disks for use

SYNOPSIS
diskparts
dmaon

DESCRIPTION
Diskparts configures FDISK and Plan 9 partitions on any disks named /dev/sd*, then configures
fs(3) by copying /cfg/$sysname/fsconfig, if present, to /dev/fs/ctl, if present, one
line at a time. If #S or #k are not bound to /dev yet, they are first bound after the current con-
tents.
Dmaon enables DMA for all attached IDE devices that claim to support it.

FILES
/dev/sd[C-H]?/ctl storage interface control files for IDE devices

SOURCE
/rc/bin

SEE ALSO
sd(3), partfs(8)
NAME
disksim – disk simulator

SYNOPSIS
aux/disksim [-r][-f file][-s srvname][-m mtpt][diskname]

DESCRIPTION
Disksim presents an in–memory disk in the manner of the sd(3) device on mtpt/diskname (default
/dev/sdXX). The disk is initialized to zeros; non-zeroed blocks written to the disk are kept in
memory.

When setting disk geometry with the geometry control message, the arguments are sectors, sector
size, cylinders, heads, and sectors per track. The last three may be zero for LBA disk simula-
tions, but must be present.

The –f option causes disksim to use file as the initial contents of the disk rather than a zeroed
image. Changes made to the disk are written back to file unless the –r option is given.

The –s option causes disksim to post its 9P service at /srv/service.

EXAMPLES
Disksim can be used to test programs such as fdisk and prep(8) that expect sd(3) disks:
    aux/disksim
echo geometry 40000 512 0 0 0 >/dev/sdXX/ctl # 20MB
disk/mbr /dev/sdXX/data
disk/fdisk -baw /dev/sdXX/data
disk/prep /dev/sdXX/plan9

Disksim is useful for creating very large but mostly zeroed files for testing other programs. Test
tar(1)'s handling of large files:
    for(i in sdXX sdYY sdZZ) aux/disksim $i
echo geometry 40000000 512 0 0 0 >/dev/sdXX/ctl # 20GB
echo geometry 10000000 512 0 0 0 >/dev/sdYY/ctl # 5GB
echo geometry 20000000 512 0 0 0 >/dev/sdZZ/ctl # 10GB
tar cf /dev/sdXX/data /dev/sdYY/data /dev/sdZZ/data
tar tvf /dev/sdXX/data

SOURCE
/sys/src/cmd/aux/disksim.c

SEE ALSO
sd(3), prep(8)
NAME
drawterm – connect to Plan 9 CPU servers from other operating systems

SYNOPSIS
drawterm [−d][−a authaddr][−c cpuaddr][−e encryption−hash−algs][−k keypattern][−s secstoreaddr][−u user][−C cmd args ...]

DESCRIPTION
Drawterm is not a Plan 9 program. It is a program that users of non–Plan 9 systems can use to establish graphical cpu(1) connections with Plan 9 CPU servers. Just as a real Plan 9 terminal does, drawterm serves its local name space as well as some devices (the keyboard, mouse, and screen) to a remote CPU server, which mounts this name space on /mnt/term and starts a shell. Typically, either explicitly or via the profile, one uses the shell to start rio(1).

By default, drawterm uses the CPU server $cpu or cpu. and the authentication server $auth or auth. The −a, −c, and −s options specify alternate authentication, CPU and secstore servers, respectively. (Edit the source to set appropriate local defaults.)

Cmd is a command to be executed remotely. Options e, k, and u have the same meaning as in cpu(1).

Drawterm has been ported to FreeBSD, Irix, Linux, NetBSD, and Windows. Binaries can be downloaded from http://swtch.com/drawterm/.

SOURCE
/sys/src/cmd/unix/drawterm

DIAGNOSTICS
Drawterm prints most diagnostics in its own window.

SEE ALSO
cpu(1), rio(1)

BUGS
Although at first drawterm may seem like a Plan 9 terminal, in fact it is just a way to provide a CPU server with some terminal devices. The difference is important because one cannot run terminal–resident programs when using drawterm. The illusion can be improved by delicate adjustments in /usr/$user/lib/profile.

It would be nice to be able to change the default servers without recompiling.
NAME
findviaduct - look up data about Viaduct bridging-and-tunnelling clients

SYNOPSIS
findviaduct ip
findviaduct -n viaduct-name

DESCRIPTION
Findviaduct prints the Viaduct name, user and bridge corresponding to an ip address, or under -n, prints bridging and tunnelling information for a named Viaduct.

FILES
/sys/log/ipboot
/adm/brickdb/owners

SOURCE
/rc/bin/findviaduct

SEE ALSO
bridge(3)
/usr/ehg/doc/viaduct
NAME
fossilcons - fossil console commands

SYNOPSIS
con /srv/fscons
  . file
  9p T-message
  bind [-b|a|-c|bc|ac ] new old
dflag
echo [-n][ arg]
listen [-INd][ address]
msg [-m nmsg ][-p nproc]
printconfig
srv [-APWdp] name
uname name [id]:id | %newname | =leader | +member | -member]
users [-d | -r file][]-w]
who

fsys name
fsys name config device
fsys name venti [ host]
fsys name open [-APVWr][-c ncache]
[fsys name] close
fsys name unconfig

[fsys name] bfree addr
[fsys name] block addr off [count [data]]
[fsys name] check [pblock][pdir][pfile][bclose][clri][clre][clrp][
  fix][venti][snapshot]
[fsys name] clre addr offsets ...
[fsys name] clri files ...
[fsys name] clrp addr offset ...
[fsys name] create path uid gid perm
[fsys name] df
[fsys name] epoch [ -ry ] n]
[fsys name] halt
[fsys name] label addr [ type state epoch epochclose tag ]
[fsys name] remove files ...
[fsys name] snap [-a][-s src][-d dst]
[fsys name] snapclean[ timeout]
[fsys name] snaptime [ -a hhmm ][-s interval][-t timeout]
[fsys name] stat files...
[fsys name] sync
[fsys name] unhalt
[fsys name] vac dir
[fsys name] wstat file elem uid gid perm length

DESCRIPTION
These are configuration and maintenance commands executed at the console of a fossil(4) file
server. The commands are split into three groups above: file server configuration, file system con-
fignation, and file system maintenance. This manual page is split in the same way.

File server configuration
The dot (.) command reads file, treating each line as a command to be executed. Blank lines and
lines beginning with a # character are ignored. Errors during execution are printed but do not
stop the script. Note that file is a file in the name space in which fossil was started, not a file in
any file system served by *fossil*.

9p executes a 9P transaction; the arguments are in the same format used by *9pcon*(8).

*Bind* behaves similarly to *bind*(1). It is useful when fossil is started without devices it needs configured into its namespace.

*Dflag* toggles the debug flag and prints the new setting. When the debug flag is set, all protocol messages and information about authentication is printed to standard *error*.

*Echo* behaves identically to *echo*(1), writing to the console.

*Listen* manages the network addresses at which fossil is listening. With no arguments, *listen* prints the current list of addresses and their network directories. With one argument, *listen* starts a new listener at *address*; the −d flag causes *listen* to remove the listener at the given address. By default, the user *none* is only allowed to attach on a connection after at least one other user has successfully attached. The −N flag allows connections from *none* at any time. The −I flag causes *fossil* to check the IP address of incoming connections against /mnt/ipok, rejecting attaches from disallowed addresses. This mechanism is not intended for general use. The server *sources.cs.bell-labs.com* uses it to comply with U.S. crytography export regulations.

*Msg* prints the maximum internal 9P message queue size and the maximum number of 9P processes to allocate for serving the queue. The −m and −p options set the two variables.

*Printconfig* prints the config line for each configured file system and prints the *venti* line, if any, used to configure this file server.

*Srv* behaves like *listen* but uses /sr"v/name* rather than a network address. With the −p flag, *srv* edits a list of console services rather than 9P services. With no arguments, *srv* prints the current list of services. With one argument, *srv name* starts a new service at /sr"v/name*; the −d flag causes *srv* to remove the named service. See the [fsys] *open* command below for a description of the −APW options.

*Uname* manipulates entries in the user table. There is no distinction between users and groups: a user is a group with one member. For each user, the user table records:

*id* the string used to represent this user in the on-disk structures

*name* the string used to represent this user in the 9P protocol

*leader* the group’s leader (see *stat*(5) for a description of the special privileges held by a group leader)

*members* a comma-separated list of members in this group

The *id* and *name* are usually the same string, but need not be. Once an *id* is used in file system structures archived to Venti, it is impossible to change those disk structures, and thus impossible to rename the *id*. The translation from *name* to *id* allows the appearance of renaming the user even though the on-disk structures still record the old name. (In a conventional Unix file system, the *id* is stored as a small integer rather than a string.) *Leader* and *members* are names, not ids.

The first argument to *uname* is the *name* of a user. The second argument is a verb, one of:

*id* create a user with name ‘*name*’ and id ‘*id*’; also create a home directory /active/usr/*uname*

:*id* create a user with name ‘*name*’ and id ‘*id*’, but do not create a home directory

*%newname* rename user ‘*name*’ to ‘*newname*,’ throughout the user table

=leader set *name*’s group leader to *leader*.

= remove *name*’s group leader; then all members will be considered leaders

+*member* add *member* to *name*’s list of members

−*member* remove *member* from *name*’s list of members
If the verb is omitted, the entire entry for *name* is printed, in the form
*‘id: name: leader: members.’*

The end of this manual page gives examples.

*Users* manipulates the user table. The user table is a list of lines in the form printed by the *uname* command. The −d flag resets the user table with the default:

```
adm:adm:adm:sys
none:none::
noworld:noworld::
sys:sys::
glenda:glenda:glenda:
```

Except glenda, these users are mandatory: they must appear in all user files and cannot be renamed.

The −r flag reads a user table from the named *file* in file system main. The −w flag writes the table to /active/adm/users on the file system main. /active/adm and /active/adm/users will be created if they do not exist.

*Users* −r /active/adm/users is automatically executed when the file system main is opened.

*Users* −w is automatically executed after each change to the user table by the *uname* command.

*Who* prints a list of users attached to each active connection.

**File system configuration**

*Fsys* sets the current file system to *name*, which must be configured and open (q.v.). The current file system name is displayed as the file server prompt. The special name *all* stands for all file systems; commands applied to *all* are applied to each file system in turn. The commands *config*, *open*, *venti*, and *close* cannot be applied to *all*.

*Fsys* takes as an optional argument (after *name*) a command to execute on the named file system. Most commands require that the named file system be configured and open; these commands can be invoked without the *fsys name* prefix, in which case the current file system is used. A few commands (*config*, *open*, and *unconfig*) operate on unopened file systems; they require the prefix.

*Config* creates a new file system named *name* using disk file *device*. This just adds an entry to fossil’s internal table.

*Venti* establishes a connection to the Venti server *host* (by default, the environment variable $venti or the network variable $venti) for use by the named file system. If *no venti* command is issued before *open*, the default Venti server will be used. If the file system is open, and was not opened with the −V flag, the command redial the Venti server. This can be used to reestablish broken connections. It is not a good idea to use the command to switch between Venti servers, since Fossil does not keep track of which blocks are stored on which servers.

*Open* opens the file system, reading the root and super blocks and allocating an in-memory cache for disk and Venti blocks. The options are:

```
-A run with no authentication
-P run with no permission checking
-V do not attempt to connect to a Venti server
-W allow wstat to make arbitrary changes to the user and group fields
-r open the file system read-only
-c ncach e allocate an in-memory cache of ncach e (by default, 1000) blocks
```

The −APW settings can be overridden on a per-connection basis by the *srv* command above.

*Close* flushes all dirty file system blocks to disk and then closes the device file.

*Unconfig* removes the named file system (which must be closed) from fossil’s internal table.
File system maintenance

Bfree marks the block at disk address addr as available for allocation. Before doing so, it prints a label command (q.v.) that can be used to restore the block to its previous state.

Block displays (in hexadecimal) the contents of the block at disk address addr, starting at offset and continuing for count bytes or until the end of the block. If data (also hexadecimal) is given, the contents in that range are replaced with data. When writing to a block, block prints the old and new contents, so that the change is easily undone. Editing blocks is discouraged.

Clre zeros an entry from a disk block. Before doing so, it prints a block command that can be used to restore the entry.

Clri removes the internal directory entry and abandons storage associated with files. It ignores the usual rules for sanity, such as checking against removing a non-empty directory. A subsequent fchek (see fossil(4)) will identify the abandoned storage so it can be reclaimed with bfree commands.

Clrp zeros a pointer in a disk block. Before doing so, it prints a block command that can be used to restore the entry.

Check checks the file system for various inconsistencies. If the file system is not already halted, it is halted for the duration of the check. If the archiver is currently sending a snapshot to Venti, the check will refuse to run; the only recourse is to wait for the archiver to finish.

A list of keyword options control the check. The pblock, pdir, and pfile options cause check to print the name of each block, directory, or file encountered.

By default, check reports errors but does not fix them. The bclose, clri, clre, and clrp options specify correcting actions that may be taken: closing leaked blocks, clearing bad file directory entries, clearing bad pointers, and clearing bad entries. The fix option enables all of these; it is equivalent to bclose clri clre clrp.

By default, check scans the portion of the active file system held in the write buffer, avoiding blocks stored on Venti or used only in snapshots. The venti option causes check to scan the portion of the file system stored on Venti, and the snapshot option causes check to scan old snapshots. Specifying snapshot causes check to take a long time; specifying venti or (worse) venti snapshot causes check to take a very long time.

Create creates a file on the current file system. Uid and gid are uids (not unames; see the discussion above, in the description of the uname command). Perm is the low 9 bits of the permission mode of the file, in octal. The a, d, and 1 mode prefixes set the append-only, directory, and lock bits. The perm is formatted as described in the stat command; creating files or directories with the snapshot(s) bit set is not allowed.

Df prints the amount of used disk space in the write buffer.

Epoch sets the low file system epoch. Snapshots in the file system are given increasing epoch numbers. The file system maintains a low and a high epoch number, and only allows access to snapshots in that range. The low epoch number can be moved forward to discard old snapshots and reclaim the disk space they occupy. (The high epoch number is always the epoch of the currently active file system.)

With no argument epoch reports the current low and high epoch numbers. The command “epoch n” is used to propose changing the low epoch to n. In response, fossil scans /archive and /snapshot for snapshots that would be discarded, printing their epoch numbers and the clri commands necessary to remove them. The epoch is changed only if no such paths are found. The usual sequence of commands is (1) run epoch to print the snapshots and their epochs, (2) clri some snapshots, (3) run epoch again. If the file system is completely full (there are no free blocks), clri may fail because it needs to allocate blocks. For this situation, the --y flag to epoch forces the epoch change even when it means discarding currently accessible snapshots. Note that when there are still snapshots in /archive, the archiver should take care of those snapshots (moving the blocks from disk to Venti) if you give it more time.

The --1 flag to epoch causes it to remove any now-inaccessible snapshot directories once it has changed the epoch. This flag only makes sense in conjunction with the --y flag.

Epoch is a very low-level way to retire snapshots. The preferred way is by setting an automatic timer with snaptime.
Halt suspends all file system activity; unhalt resumes activity.

Label displays and edits the label associated with a block. When editing, a parameter of − means leave that field unchanged. Editing labels is discouraged.

Remove removes files.

Snap takes a temporary snapshot of the current file system, recording it in /snapshot/yyyy/mm/dd/hh:mm as described in fossil(4). The −a flag causes snap to take an archival snapshot, recording it in /archive/yyyy/mm/dd, also described in fossil(4). By default the snapshot is taken of /active, the root of the active file system. The −s flag specifies a different source path. The −d flag specifies a different destination path. These two flags are useful together for moving snapshots into the archive tree.

Snapclean immediately discards all snapshots that are more than timeout minutes old. The default timeout is the one set by the snaptime command. The discarding is a one-time event rather than a recurring event as in snaptime.

Snaptime displays and edits the times at which snapshots are automatically taken. An archival snapshot is taken once a day, at hh:mm, while temporary snapshots are taken at multiples of interval minutes. Temporary snapshots are discarded after they are timeout minutes old. The snapshot cleanup runs every timeout minutes or once a day, whichever is more frequent, so snapshots may grow to an age of almost twice the timeout before actually being discarded. With no arguments, snaptime prints the current snapshot times. The −a and −s options set the archive and snapshot times. An hh:mm or interval of none can be used to disable that kind of automatic snapshot. The −t option sets the snapshot timeout. If timeout is none, temporary snapshots are not automatically discarded. By default, all three times are set to none.

Stat displays metadata for each of the named files, in the form:

```
stat file elem uid gid perm length
```

(Replacing stat with wstat yields a valid command.) The perm is an octal number less than or equal to 777, prefixed with any of the following letters to indicate additional bits.

- a append only
- d directory
- l exclusive use
- s is the root of a snapshot
- t temporary bit
- A MS-DOS archive bit
- G setgid
- H MS-DOS hidden bit
- L symbolic link
- S MS-DOS system bit
- U setuid
- Y sticky

The bits denoted by capital letters are included to support non–Plan 9 systems. They are not made visible by the 9P protocol.

Sync writes dirty blocks in memory to the disk.

Vac prints the Venti score for a vac(1) archive containing the tree rooted at dir, which must already be archived to Venti (typically dir is a directory in the /archive tree).

Wstat changes the metadata of the named file. Specifying − for any of the fields means “don’t change.” Attempts to change the d or s bits in the perm are silently ignored.

**EXAMPLES**

Sources, the Plan 9 distribution file server, uses the following configuration file:

```
srv −p fscons.sources
srv −p fscons.sources.adduserd
srv sources
fsys main config /dev/sdC0/fossil.outside
fsys main open −c 25600
fsys main
```
users /active/adm/users
listen tcp!*:564
msg -m 40 -p 10
   snaptime -a 0000 -s 15

The second console is used by the daemon that creates new accounts.

To add a new user with name and id rob and create his home directory:
   uname rob rob

To create a new group sys (with no home directory) and add rob to it:
   uname sys :sys
   uname sys +rob

To save an old (but not yet discarded) snapshot into the archive tree:
   snap -a -s /snapshot/2003/1220/0700 -d /archive/2003/1220
NAME
    fs, exsort - file server maintenance

SYNOPSIS
    help [command ...]
    allow
    arp subcommand
cfs filesystem
    check [options]
clean file [bno [addr]]
cli [file...]
cpu [proc]
create path uid gid perm [lad]
cwcmd subcommand
date [+-] seconds
disallow
duallow [uid]
dump [filesystem]
files
flag flag [channel]
fsstat [files]
halt
hangup channel
newuser name [options]
noattach
passwd
printconf
profile [01]
remove [files...]
route subcommand
snntp kick
stat[admiesw]
stats [+-] flags...
sync
time command
trace [number]
users [file]
version
who [user...]
wormeject [tunit]
wormingest [tunit]
wormoffline drive
wormonline drive
wormreset
disk/exsort [-w] [file]

DESCRIPTION
    Except for exsort, these commands are available only on the console of an fs(4) file server.
Help prints a 'usage string' for the named commands, by default all commands. Also, many commands print menus of their options if given incorrect or incomplete parameters.
Allow disables permission checking and allows wstat. This may help in initializing a file system. Use this with caution.
Arp has two subcommands: print prints the contents of the ARP cache and flush flushes it.
Cfs changes the current file system, that is, the file tree to which commands (check, clean, cli, create, cwcmd, dump, newuser, profile, remove, and users) apply. The initial filesystem is main.
Check verifies the consistency of the current file system. With no options it checks and reports the status. It suspends service while running. Options are:

- **rdall**: Read every block in the file system (can take a long time). Normally, check will stop short of the actual contents of a file and just verify the block addresses.
- **tag**: Fix bad tags; each block has a tag that acts as a backwards pointer for consistency checking.
- **ream**: Fix bad tags and also clear the contents of blocks that have bad tags.
- **pfile**: Print every file name.
- **pdir**: Print every directory name.
- **free**: Rebuild the list of free blocks with all blocks that are not referenced. This option is only useful on non-cache/WORM file systems. If the filesystem was modified, the summary printed at the conclusion of the check may not reflect the true state of the freelist and may also print a list of missing blocks. These missing blocks are actually on the free list and the true state of the filesystem can be determined by running check with no arguments.
- **bad**: Each block address that is out of range or duplicate is cleared. Note that only the second and subsequent use of a block is cleared. Often the problems in a file system are caused by one bad file that has a lot of garbage block addresses. In such a case, it is wiser to use check to find the bad file (by number of diagnostic messages) and then use clri to clear the addresses in that file. After that, check can be used to reclaim the free list.
- **touch**: Cause every directory and indirect block not on the current WORM disk to be advanced to the current WORM on the next dump. This is a discredited idea to try to keep operating on the knee of the cache working set. Buy more cache disk.
- **trim**: Reduces the file system’s fsize to fit the device containing the file system. This is useful after copying a partially-full file system into a slightly smaller device. Running check free afterward will construct a new free list that contains no blocks outside the new, smaller file system.

Clean prints the block numbers in file’s directory entry (direct, indirect and doubly indirect) and checks the tags of the blocks cited. If bno is supplied, the bno’th block number (using zero origin) is set to addr (defaults to zero). Note that only the block numbers in the directory entry itself are examined; clean does not recurse through indirect blocks.

Clri clears the internal directory entry and abandons storage associated with files. It ignores the usual rules for sanity, such as checking against removing a non-empty directory. A subsequent check free will place the abandoned storage in the free list.

Cpu prints the CPU utilization and state of the processes in the file server. If the name of a process type argument is given, then CPU utilization for only those processes is printed.

Create creates a file on the current file system. Uid and gid are names or numbers from /adm/users. Perm is the low 9 bits of the permission mode of the file, in octal. An optional final l, a, or d creates a locked file, append-only file, or directory.

Cwcmd controls the cached WORM file systems, specifically the current file system. The subcommands are:

- **mvstate state1 state2 [ platter]**
  States are none, dirty, dump, dump1, error, read, and write. A mvstate dump1 dump will cause I/O errors in the last dump to be retried. A mvstate dump1 write will cause I/O errors in the last dump to be retried in reallocated slots in the next dump. A mvstate read none will flush the cache associated with the WORM. A mvstate dump write aborts the background process dumping to WORM; as a consequence it leaves holes in the dump file system. Other uses are possible but arcane. The optional platter limits affected blocks to those on that platter.

- **prchain [start] [back–flag]**
  Print the chain of superblocks for the directory containing the roots of the dumped file systems, starting at block number start (default 0) going forward (backwards if back–flag is
supplied and is non-zero).

**searchtag** [start] [tag] [blocks]
Reads the WORM device starting at block start and proceeding for blocks blocks (default 1000) until it finds a block with numeric tag tag.

**savecache** [percent]
Copy the block numbers, in native endian longwords, of blocks in the read state to the file /adm/cache for use by disk/exsort. If an argument is given, then that percent (most recently used) of each cache bucket is copied.

**loadcache** [dskno]
Read /adm/cache and for every block there on WORM disk side dskno (zero-origin), read the block from WORM to the cache. If dskno is not supplied, all blocks in /adm/cache are read.

**morecache** dskno [count]
Read count blocks from the beginning of WORM disk side dskno to the cache. If no count is given, read all of side dskno into the cache.

**startdump** [01]
Suspend (0) or restart (1) the background dump process.

**touchsb**
Verify that the superblock on the WORM is readable, ignoring the cached copy.

**blockcmp** [wbnol] [cbno]
Compares the WORM block wbnol with the cache block cbno and prints the first 10 differences, if any.

**acct**
Prints how many times each user has used the system to allocate new space on the WORM; the units are megabytes.

**clearacct**
Clears the accounting records for acct.

**Date**
prints the current date. It may be adjusted using +-- seconds. With no sign, it sets the date to the absolute number of seconds since 00:00 Jan 1, 1970 GMT; with a sign it trims the current time.

**Disallow**
restores permission checking back to normal after a file system has been initialized.

**Dualow**
sets permissions such that the named user can read and search any directories. This is the permission necessary to do a du(1) command anywhere in the file system to discover disk usage.

**Dump**
starts a dump to WORM immediately for the named filesystem, or the current filesystem if none is named. File service is suspended while the cache is scanned; service resumes when the copy to WORM starts.

**Files**
prints for every connection the number of allocated fids.

**Fstat**
prints the current status of each named file, including uid, gid, wuid (uid of the last user to modify the file), size, qid, and disk addresses.

**Flag**
Toggles flags, initially all off:

- **allchans**
  Print channels in who output.
- **arp**
  Report ARP activity.
- **attach**
  Report as connections are made to the file server.
- **authdebug**
  Report authentications.
- **authdisable**
  Disable authentication.
- **chat**
  (Very noisy.) Print all 9P messages to and from the server.
- **error**
  Report 9P errors.
- **il**
  Report IL errors.
- **route**
  Report received RIP packets.
- **ro**
  Report I/O on the WORM device.
- **sntp**
  Report SNTP activity.

If given a second numeric channel argument, as reported by who, the flag is altered only on that connection.
**Halt** does a **sync** and halts the machine, returning to the boot ROM.

**Hangup** clunks all the fids on the named **channel**, which has the same format as in the output of the **who** command.

**Newuser** requires a **name** argument. With no options it adds user **name**, with group leader **name**, to /adm/users and makes the directory /usr/name owned by user and group **name**. The options are

```
?         Print the entry for **name**.
:        Add a group: add the name to /adm/users but don’t create the directory. By
        convention, groups are numbered starting from 10000, users from 0.
newname  Rename existing user **name** to **newname**.
=leader  Change the leader of **name** to **leader**. If **leader** is missing, remove the existing
        leader.
+member  Add **member** to the member list of **name**.
−member  Remove existing **member** from the member list of **name**.
```

After a successful **newuser** command the file server overwrites /adm/users to reflect the internal state of the user table.

**Noattach** disables **attach**(5) messages, in particular for system maintenance. Previously attached connections are unaffected. Another **noattach** will enable normal behavior.

**Passwd** sets the machine’s password and writes it in non–volatile RAM.

**Printconf** prints the system configuration information.

**Profile** 1 clears the profiling buffer and enables profiling; **profile** 0 stops profiling and writes the data to /adm/kprofdata for use by kprof (see **prof**(1)). If a number is not specified, the profiling state toggles.

**Remove** removes files.

**Route** maintains an IP routing table. The **subcommands** are:

```
add  dest gate [mask]     Add a static route from IP address **dest** using gateway **gate** with an
                          optional subnet **mask**.
delete dest               Delete an entry from the routing table.
print                      Display the contents of the routing table.
ripon                      Enables the table to be filled from RIP packets.
ripooff                    Disables the table from being updated by RIP packets.
```

**Sntp kick** queries the SNTP server (see **fsconfig**(8)) and sets the time with its response.

The **stat** commands are connected with a service or device identified by the last character of the name: d, SCSI targets; e, Ethernet controllers; i, IDE/ATA targets; m, Marvell SATA targets; w, cached WORM. The **stata** command prints overall statistics about the file system. The **stats** command takes an optional argument identifying the characters of **stat** commands to run. The option is remembered and becomes the default for subsequent **stats** commands if it begins with a minus sign.

**Sync** writes dirty blocks in memory to the magnetic disk cache.

**Time** reports the time required to execute the **command**.

**Trace** with no options prints the set of queue–locks held by each process in the file server. If things are quiescent, there should be no output. With an argument **number** it prints a stack traceback of that process.

**Users** uses the contents of **file** (default /adm/users) to initialize the file server’s internal representation of the users structure. Incorrectly formatted entries in **file** will be ignored. If file is explicitly **default**, the system builds a minimal functional users table internally; this can help recover from disasters. If the **file** cannot be read, you **must** run

```
users default
```

for the system to function. The **default** table looks like this:

```
−1:adm:adm:
   0:none:adm:
```
Version reports when the file server was last compiled and last rebooted.

Who reports, one per line, the names of users connected to the file server and the status of their connections. The first number printed on each line is the channel number of the connection. If users are given the output selects connections owned by those users.

Wormeject moves the WORM disk in slot tunit of the first jukebox to the output shelf.

Wormingest moves the WORM disk from the input shelf of the first jukebox to slot tunit.

Wormoffline takes drive of the first jukebox out of service; wormonline puts it back in service.

Wormreset put discs back where the jukebox thinks they belong, and does this for all jukeboxes.

When the file server boots, it prints the message
  
  for config mode hit a key within 5 seconds
  
If a character is typed within 5 seconds of the message appearing, the server will enter config mode. See fsconfig(8) for the commands available in config mode. The system also enters config mode if, at boot time, the non-volatile RAM does not appear to contain a valid configuration.

Exsort is a regular command to be run on a CPU server, not on the file server console. It reads the named file (default /adm/cache) and sorts the cache disk block numbers contained therein. It assumes the numbers are 4-byte integers and guesses the endianness by looking at the data. It then prints statistics about the cache. With option -w it writes the sorted data back to file.

SEE ALSO

fs(4)

Ken Thompson, “The Plan 9 File Server”.

SOURCE

/sys/src/fs
/sys/src/cmd/disk/exsort.c

BUGS

The worm* commands should accept an argument identifying a jukebox.
NAME
fsconfig - configuring a file server

SYNOPSIS
service name
config device
nvram device
filsys name device
ip ipaddr
ipgw ipaddr
ipmask ipaddr
ipauth ipaddr
ipsntp ipaddr
ream name
recover name
allow
readonly
noauth
noattach
copyworm
copydev from-dev to-dev
halt
des
end

DESCRIPTION
When an fs(4) file server’s configuration has not been set, or by explicit request early in the server’s initialization (see fs(8)), the server enters ‘config mode’. The commands described here apply only in that mode. They establish configuration constants that are typically valid for the life of the server, and therefore need be run only once. If the non-volatile RAM on the server gets erased, it will be necessary to recreate the configuration.

Syntax
In these commands, ipaddr is an IP address in the form 111.103.94.19 and name is a text string without white space. The syntax of a device is more complicated:

wn1.n2.n3
Defines a SCSI disk on target (unit) id n2, controller (host adapter) n1, and LUN (logical unit number) n3. A single number specifies a target, while two numbers specify target.lun, with the missing numbers defaulting to zero. Any one of the numbers may be replaced by <m–n> to represent the values m through n inclusive. M may be greater than n. For example, (w<1–4>) is the concatenation of SCSI targets 1 through 4.

hn1.n2.n3
H is similar to w, but for IDE or ATA disks, and the controllers must be specified in plan9.ini. Lun is ignored. Target 0 is an IDE master and 1 is a slave. Instead of specifying controller and target separately, one may omit the controller and specify a target of controller–number*2 + target–number, thus h2 is equivalent to h1.0.0 (second IDE controller, master drive).

mn1.n2.n3
M is similar to h, but for SATA drives connected to Marvell 88SX[56]0[48][01] controllers. There is no need to specify the controllers in plan9.ini as they are autodiscovered. Hot-swapping drives is not currently supported. Similar target naming rules apply as for IDE controllers. However the controller–number is multiplied by the number of drives the
controller supports rather than 2. Thus m9 is equivalent to m1.1.0 (second controller, second drive), if the first controller supports 8 drives.

\ln1.n2.n3
\ln1.n2.n3

The same as \w, but leaving a single block at the beginning for a label (\l), or not. Only \n2 is really of interest, and refers to a side of a WORM disc. These are only really relevant when used as \device3 in the \j device (see below).

\(\text{\device...}\)
A pseudo–device formed from the concatenation of the \devices in the list. The devices are not blank– or comma–separated.

[ \text{\device...} ]
A pseudo–device formed from the block-wise interleaving of the \devices in the list. The size of the result is the number of devices times the size of the smallest device.

{ \text{\device...} }
A pseudo–device formed from the mirroring of the first \device in the list onto all the others. The size of the result is the size of the smallest device. One might think of this as RAID 1, and [ ] as RAID 0, though neither includes any fancy recovery mechanisms. Each block is written to all the devices, starting with the rightmost in the list and working leftward. A block is read from the first device that provides it without error, starting with the leftmost in the list and working rightward.

\pdevice.n1.n2
A partition starting at \n1\% from the beginning of \device with a length \n2\% of the size of the device. Parenthesize \device if it contains periods.

\xdevice
A pseudo–device that contains the byte-swapped contents of \device. Since the file server writes integers to disk in its native byte order, it can be necessary to use this device to read file systems written by processors of the other byte order.

\j(device1 device2...\device3)
Device1 is the SCSI juke box interface. The devices are the SCSI drives in the jukebox and \device3 represents the demountable platters in the juke box.

\fdevice
A pseudo–WORM disk: blocks on \device can be written only once and may not be read unless written.

\cdevice1 device2
A cached WORM. The first \device is the cache, the second the WORM.

\o
(Letter o) The read-only (dump) file system of the most–recently defined cached WORM file system.

**Configuration**

The service command sets the textual name of the server as known in the network databases.

The configuration information is stored in block zero on a device whose device string is written in non–volatile RAM. The config and nvramp commands identify the \device on which the information is recorded. The config command also erases any previous configuration.

The filesys command configures a file system on \device and calls it name. Name is used as the specifier in attach messages to connect to that file system. (The file system main is the one attached to if the specifier is null; see attach(5)).

The rest of the configuration commands record IP addresses: the file server’s address (ip), the local gateway’s (ipgw), the local authentication server’s (ipauth), the local subnet mask (ipmask), and the address of a system running an SNTP server (ipsntp). Ipauth is no longer used. If the server has more than one network interface, a digit may be appended to the keywords ip, ipgw and ipmask to indicate the interface number; zero is the default.

**One–time actions**

The ream command initializes the named file system. It overwrites any previous file system on the same device and creates an empty root directory on the device. If name is main, the file server,
until the next reboot, will accept wstat messages (see stat(5)) that change the owner and group of files, to enable initializing a fresh file system from a mkfs(8) archive.

For the recover command, the named file system must be a cached WORM. Recover clears the associated magnetic cache and initializes the file system, effectively resetting its contents to the last dump.

Allow turns off all permission checking; use with caution.

ReadOnly disables all writing to all devices. This is useful for trying dangerous experiments.

Noauth disables authentication.

Noattach prevents attaches.

Copyworm will copy a file system named main to one named output, block by block, and loop. It knows how to read a fake worm file system.

Copydev will copy the device from−dev to the device to−dev, block by block, and panic.

Halt will cause the server to immediately exit and reboot.

The various configuration commands only record what to do; they write no data to disk. The command end exits config mode and begins running the file server proper. The server will then perform whatever I/O is required to establish the configuration.

EXAMPLE

Initialize a file server kgbsun with a single file system interleaved between SCSI targets 3 and 4.

```
service kgbsun
config w3
 filsys main [w<3–4>]
 ream main
```

Initialize a file server kremvax with a single disk on target 0 partitioned as a cached pseudo−WORM file system with the cache on the third quarter of the drive and the pseudo−WORM on the interleave of the first, second, and fourth quarters.

```
service kremvax
config p(w0)50.1
 filsys main cp(w0)50.25f[p(w0)0.25p(w0)25.25p(w0)75.25]
 filsys dump o
 ream main
```

A complete and complex example: initialize a file server fsb with a single SCSI disk on target 0 for a scratch file system, a cached WORM file system with cache disk on target 2 and an optical−disc jukebox on targets 4 (robotics) and 5 (one optical drive), and another cached WORM file system with cache disk on target 3 and another optical−disc jukebox on a second SCSI bus at targets 3 and 4. Both jukeboxes contain 16 slots of optical discs. It has two Ethernet interfaces and can reach an SNTP server on the first one.

```
service fsb
config w0
 filsys main cw2j(w4w5)(l<0–31>)
 filsys dump o
 filsys hp40fx cw3j(w1.<3–4>.0)(l<0–31>)
 filsys hp40fxdump o
 filsys other w0
 ipauth 0.0.0.0
 ipsnntp 10.9.0.3
 ip0 10.9.0.2
 ipgw0 10.9.0.3
 ipmask0 255.255.0.0
 ip1 10.0.0.2
 ipgw1 10.0.0.1
 ipmask1 255.255.0.0
 ream main
 ream hp40fx
```
ream other
end

SOURCE
/sys/src/fs/port/config.c

SEE ALSO
Ken Thompson, “The Plan 9 File Server”. 
NAME
    fshalt, reboot - halt any local file systems and optionally reboot the system

SYNOPSIS
    fshalt [-r]
    reboot

DESCRIPTION
    Fshalt syncs all local fossil(4), venti(8), and kfs(4) servers, then halts all local fossil and kfs
    servers. If given -r, fshalt will then reboot the machine. The halting and rebooting is done by
    copying all necessary commands into a ramfs(4) file system and changing directory there before
    attempting to halt file systems, so this will work even on standalone machines with their roots on
    local file systems.

    Reboot restarts the machine it is invoked on.

SOURCE
    /rc/bin/fshalt
    /rc/bin/reboot

SEE ALSO
    cons(3), reboot(8)

BUGS
    On standalone machines, it will be impossible to do anything after invoking bare fshalt.
NAME
getflags, usage - command-line parsing for shell scripts

SYNOPSIS
aux/getflags $*
aux/usage

DESCRIPTION
Getflags parses the options in its command-line arguments according to the environment variable $flagfmt. This variable should be a list of comma-separated options. Each option can be a single letter, indicating that it does not take arguments, or a letter followed by the space-separated names of its arguments. Getflags prints an rc(1) script on standard output which initializes the environment variable $flagx for every option mentioned in $flagfmt. If the option is not present on the command-line, the script sets that option’s flag variable to an empty list. Otherwise, the script sets that option’s flag variable with a list containing the option’s arguments or, if the option takes no arguments, with the string 1. The script also sets the variable $* to the list of arguments following the options. The final line in the script sets the $status variable, to the empty string on success and to the string usage when there is an error parsing the command line.

Usage prints a usage message to standard error. It creates the message using $flagfmt, as described above, $args, which should contain the string to be printed explaining non-option arguments, and $0, the program name (see rc(1)).

EXAMPLE
Parse the arguments for leak(1):

flagfmt='b,s,f binary,r res,x width'
args='name | pid list'
if(! ifs=() eval '{aux/getflags $*} || ~ $#* 0){
    aux/usage
    exit usage
}
NAME
gpsfs, gpsevermore - GPS time and position service

SYNOPSIS
aux/gpsfs [ -d device ] [ -b baud ] [ -s srvname ] [ -m mntpt ]
aux/gpsevermore [ -d device ] [ -b baud ] [ -n baud ] [ -l location ]

DESCRIPTION
Aux/gpsfs reads an NMEA-compatible serial GPS (Global Positioning System) device and provides time and position through a file system, by default mounted on /mnt and implementing /mnt/gps.
It implements four files in the gps directory: position, time, satellites, and raw.
The read-only position file contains one line of information in 9 tab-separated fields:

fix quality 0 means position data invalid, 1 means a 2D position is available, 2 means a 3D position is available. The value is 8, 9, or 10, respectively, when the fix data comes from a file rather than an actual GPS.
zulu time universal coordinated time encoded as hhmmss followed by the character 'Z'.
system time time and date converted to the format of time(2).
longitude in degrees, east of Greenwich is positive, west negative.
latitude in degrees, positive is north, negative south of the equator.
alitude above sea level, in meters.
course degrees, clockwise from true north.
ground speed in km/h

magnetic deviation (not provided by all GPSs), in degrees, positive is westerly, negative easterly.
The read-only time file contains one line of information in 4 tab-separated fields:
gps time in time(2) format.
gps time in nsec (see time(2)) format (ms accuracy).
system time in nsec format. This is the system time at the time of the gps time sample. The difference between this and the previous field is used in clock synchronization. See timesync(8).
validity the character A meaning sample valid and usable for clock synchronization. The other values are not usable for clock sync: B means valid sample from file playback, V means invalid sample, and W means invalid playback sample.
The read-only satellites file contains information about the current satellite constellation. It consists of one line of general information, followed by zero or more lines, one for each satellite in use. The first line contains two fields:

fix quality same as in the position file.
satellites in view number of satellites above the horizon
Subsequent lines have four fields:

prn satellite ID
elevation above the horizon, degrees.
azimuth direction, degrees from true north
snr Signal to noise ratio, 0 – 99 dB

The contents of these files are refreshed once per second when reading from an actual GPS, and once per 100 ms (giving a speed up of a factor 10) when playing back from file.
The read-only raw file can be read to obtain a copy of the raw NMEA GPS output. Gpsfs keeps an internal buffer of 8KB, so the reader must keep up with the output (typically 500 or so bytes per
second).

The -d flag establishes the device the GPS samples are read from. If the device file is not a serial interface, gpsfs assumes playback from file and modifies quality parameters as such.

The -b flag specifies the baud rate of the serial line. The standard baud rate for NMEA GPS is 4800 baud, but many device allow changing to higher speeds.

The -s flag specifies the name under which the gpsfs service is posted in /srv.

The -m flag specifies a mount mount other than /mnt.

**Evermore**

Aux/gpsevermore is used to configure GPSs using an Evermore chipset.

The -d flag specifies the serial device to the GPS.

The -b flag specifies the baud rate of the serial line. The standard baud rate for NMEA GPS is 4800 baud, but many device allow changing to higher speeds.

The -n flag specifies the speed to set the GPS to. When the command finishes, the GPS should be read (and configured) at the new speed.

The -l flag is sued to specify the location to initialize the GPS to. The format is dd:mm:ssX or dd:mm:mmmX or dd.dddX, where dd stands for degrees (one or more digits), mm for minutes and ss for seconds of arc. X is one of W, E, N or S. Longitudes come with W or E, latitudes with N or S. The -l flag is followed by two such fields, one for longitude, one for latitude. They may be given in a single argument (separated by white space), or in two arguments, in either order. Initialization time is taken from time(2).

**SEE ALSO**

timesync(8), time(2)

**FILES**

/mnt/gps/position position, time, speed and heading
/mnt/gps/satellites satellites in view
/mnt/gps/time GPS time (millisecond accuracy)
/dev/eia0 default GPS device

**SOURCE**

/sys/src/cmd/aux/gps
NAME
    histogram - draw a histogram

SYNOPSIS
    histogram [-h][ -c index][ -r minx, miny, maxx, maxy][ -s scale][ -t title][ -v maxv]

DESCRIPTION
    Histogram reads numbers, one per line, from its standard input and draws them as bars in a histo-
    gram.
    Use -c to set the color index for the graph. A modulus operation on the value keeps the color
    index within the available range.
    Unless -h (hold) is given, histogram will exit when it reaches the end-of-file. It will exit immedi-
    ately if it is interrupted or if the exit menu option is chosen.
    -r sets the initial window rectangle coordinates.
    -s sets the scaling factor.
    -t sets the title displayed on a line above the histogram. The last value read is displayed to the
    right of the title.
    -v sets the maximum value that can be expected.

EXAMPLE
    Plot a sine wave:
        hoc -e 'for(i=0.0;i<20*PI;i=i+0.1) print (10+10*sin(i)), "\n"' |
        histogram -t 'sin(t), 0 ≤ t ≤ 20π' -v 20 -h
    Show the Dow Jones adjusted daily closing price back to January 1, 2000:
        site=http://ichart.finance.yahoo.com
        hget $site'/table.csv?s=DJI&a=00&b=1&c=2000' |
        awk -F, '{print $NF}' | histogram -t DJI -v 15000 -h

SOURCE
    /sys/src/cmd/histogram.c

SEE ALSO
    statusbar(8)
NAME
httpd, save, imagemap, man2html, webls - HTTP server

SYNOPSIS
ip/httpd/httpd [-a srvaddr] [-c cert [-C certchain]] [-d domain] [-n namespace] [-w webroot]
version uri [search]
ip/httpd/imagemap ...
ip/httpd/man2html ...
ip/httpd/webls ...

DESCRIPTION
Httpd serves the webroot directory of the file system described by namespace (default
/lib/namespace.httpd), using version 1.1 of the HTTP protocol. It announces the service
srvaddr (default tcp!http), and listens for incoming calls. If an X.509 certificate is supplied
with the -c option, then the service is instead tcp!https. There should already be a facto-
tum holding the corresponding private key. If the specified certificate has been signed by a certifi-
cate authority, the -C option may be used to specify a file containing a chain of signed certificates.

Httpd supports only the GET and HEAD methods of the HTTP protocol; some magic programs sup-
port POST as well. Persistent connections are supported for HTTP/1.1 or later clients; all connec-
tions close after a magic command is executed. The Content-type (default application/octet-stream) and Content-encoding (default binary) of a file are deter-
dined by looking for suffixes of the file name in /sys/lib/mimetype.

Redirection
Each requested URI is looked up in a redirection table, read from /sys/lib/httpd.rewrite. Fields are separated by spaces and tabs. Anything following a # is ignored. The first field of each line is a URI; the second a replacement path. If a prefix of the URI matches a redirection path, the URI is rewritten using the corresponding replacement path instead of the prefix, and a temporary redirect is sent to the HTTP client. If the replacement path does not specify a server name, and the request has no explicit host, then domain is the host name used in the redirection. The prefix can either be a domain root like http://system/ (which matches that URL only) or a path like
/who/rob (which matches that path no matter what the requested server), but not both: http://system/who/rob will never match a request. If the first field ends in a slash, this is an exact match; otherwise it is a prefix match. The first field is a literal string, matched against each file prefix of each URI. The most specific, i.e., longest, pattern wins, and is applied once (there is no rescanning), except for the following exceptions. Httpd matches only the prefix and not subordinate pages if a replacement is prefixed with >. Httpd omits the unmatched part of the original URI from the rewritten URI if the replacement is prefixed with * . This permits many-to-
one mappings; for example, to send all references to an old subtree to a single error page.

Httpd handles replacements prefixed with @ internally, treating the request as if it were for the replacement (without the @) but not informing the client of the rewritten name. Replacement URLs prefixed with = generate a permanent redirection instead of a temporary one. Httpd checks to see if this file has changed once every 50 new TCP connections. HTTP 1.1 persistent connection implies many pages may come in one browser connection, so to kick-start httpd, try

for(i in '{seq 50}) hget http://www.your-domain.com/ >/dev/null

Access Control
Before opening any file, httpd looks for a file in the same directory called .httplogin. If the file exists, the directory is considered locked and the client must specify a user name and password
matching a pair in the file. .httplogin contains a list of space or newline separated tokens, each possibly delimited by single quotes. The first is a domain name presented to the HTTP client. The rest are pairs of user name and password. Thus, there can be many user name/password pairs valid for a directory.

Auxiliaries (magic)
If the requested URI begins with /magic/server/, httpd executes the file
/bin/ip/httpd/server to finish servicing the request. All the auxiliaries take the same
arguments. Method and version are those received on the first line of the request. Uri is the remaining portion of the requested URI. Inbuf contains the rest of the bytes read by the server, and netdir is the network directory for the connection. There are routines for processing command arguments, parsing headers, etc. in the httpd library, /sys/src/cmd/ip/httpd/libhttpd.a. See httpd.h in that directory and existing magic commands for more details.

Save writes a line to /usr/web/save/uri.data and returns the contents of /usr/web/save/uri.html. Both files must be accessible for the request to succeed. The saved line includes the current time and either the search string from a HEAD or GET or the first line of the body from a POST. It is used to record form submissions.

Imagemap processes an HTML imagemap query. It looks up the point search in the image map file given by uri, and returns a redirection to the appropriate page. The map file defaults to NCSA format. Any entries after a line starting with the word #cern are interpreted in CERN format.

Man2html converts man(6) format manual pages into html. It includes some abilities to search the manuals.

Webls produces directory listings on the fly, with output in the style of ls(1). /sys/lib/webls.allowed and /sys/lib/webls.denied contain regular expressions describing what parts of httpd’s namespace may and may not be listed, respectively. Webls.denied is first searched to see if access is by default denied. If so webls.allowed is then searched to see if access is explicitly allowed. Thus one can have very general expressions in the denied list (like .*), yet still allow exceptions. If webls.denied does not exist or is unreadable, all accesses are assumed to be denied unless explicitly allowed in webls.allowed.

Other sites will note that if neither webls.denied nor webls.allowed exist, any portion of httpd’s namespace can be listed (however, webls will always endeavor to prevent listing of ‘.’ and ‘.’). If webls.allowed exists but webls.denied does not, any directory to be listed must be described by a regular expression in webls.allowed. Similarly, if webls.denied exists but webls.allowed does not, any directory to be listed must not be described by a regular expression in webls.denied. If both exist, a directory is listable if either it doesn’t appear in webls.denied, or it appears in both webls.denied and webls.allowed. In other words, webls.allowed overrides webls.denied. If a listing for a directory is requested and access is denied, or another error occurs, a simple error page is returned.

EXAMPLES
These are all examples of how to use httpd.rewrite.

A local redirection:

Redirection to another site:
/netlib/lapack/lawns =http://netlib.org/lapack/lawns

Root directory for virtual host:
http://www.ampl.com /cm/cs/what/ampl

FILES
/sys/lib/mimetype content type description file
/lib/namespace.httpd default namespace file for httpd
/sys/lib/httpd.rewrite redirection file
/sys/lib/webls.allowed regular expressions describing explicitly listable pathnames; overrides webls.denied
/sys/lib/webls.denied regular expressions describing explicitly unlistable pathnames

SOURCE
/sys/src/cmd/ip/httpd

SEE ALSO
newns in auth(2), listen(8), rsa(8)
NAME
init - initialize machine upon booting

SYNOPSIS
/$cputype/init [−ctm][ command ...]

DESCRIPTION
Init initializes the machine: it establishes the name space (see namespace(4) and newns in auth(2)), and environment (see env(3)) and starts a shell (rc(1)) on the console. If a command is supplied, that is run instead of the shell. On a CPU server the invoked shell runs cpurc(8) before accepting commands on the console; on a terminal, it runs termrc and then the user’s profile. Options −t (terminal) and −c (CPU) force the behavior to correspond to the specified service class. Otherwise init uses the value of the environment variable $service to decide the service class.

Init sets environment variables $service (either to the incoming value or according to −t or −c), $objtype (to the value of $cputype), $user (to the contents of #c/user), and $timezone (to the contents of /adm/timezone/local).

With option −m init starts only an interactive shell regardless of the command or service class.

On a CPU server, init requires the machine’s password to be supplied before starting rc on the console.

Init is invoked by boot(8), which sets the arguments as appropriate.

SOURCE
/sys/src/cmd/init.c

SEE ALSO
rc(1), auth(2), boot(8)
NAME
ipconfig, rip, linklocal, ipv6on - Internet configuration and routing

SYNOPSIS
ip/ipconfig [-6DGNOPdnpzuX] [-b baud] [-c ctl] [-g gateway] [-h host] [-m mtu] [-o
dhcp-opts] [-x netmtpt] [-t gwipv4] [verb] [local] [mask] [remote] [file-server] [auth]]

ip/rip [-bdr] [-x netmtpt]
ip/linklocal [-t gwipv4] mac ...
ip6on [netmtpt ndbfile [gwv4]]

DESCRIPTION
Ipconfig binds a device interface (default /net/ether0) to a mounted IP stack (default /net) and
figures the interface with a local address and optionally a mask, a remote address, a file
server and an authentication server address. The addresses can be specified in the command line
or obtained via DHCP. If DHCP is requested, it will also obtain the addresses of DNS servers, NTP
servers, gateways, a Plan 9 file server, and a Plan 9 authentication server. If this is the first non-
loopback interface on the IP stack, the information will be written to /net/ndb in the form of an
ndb(8) entry.

Type may be ether, gbe, ppp, pkt, or loopback. The gbe type is equivalent to ether
except that it allows jumbo packets (up to ~9KB). The pkt interface passes all IP packets to and
from a user program. For ppp the device can be any byte stream device.

The verb (default add) determines the action performed. The usual verbs are:

add if the device is not bound to the IP stack, bind it. Add the given local address, mask,
and remote address to the interface. An interface may have multiple addresses.

remove remove the address from the device interface.
unbind unbind the device interface and all its addresses from the IP stack.

The IPv6-specific verbs, which take different arguments, are:

add6 prefix pfx-len onlink auto validlt prefix
sets the named IPv6 parameters; see ip(3) for more detail.

ra6 [keyword value] ...
sets IPv6 router advertisement parameter keyword's value. See ip(3) for more detail. Setting
recvra non-zero also forks a process to receive and process router advertisements. Setting
sendra non-zero also enables IP routing on the interface, forks a process to send
router advertisements, and if no recvra process is running, forks one.

The options are:

6 if adding an address (the default action), add the IPv6 link-local address.
b the baud rate to use on a serial line when configuring PPP.
c write the control string ctl to the ethernet device control file before starting to configure it.
May be repeated to specify multiple control writes.
d use DHCP to determine any unspecified configuration parameters.
d turn on debugging.
g the default gateway.
G use only generic DHCP options. Without this option, ipconfig adds to requests a Vendor Class
option with value plan9_$cpu_type and also requests vendor specific options 128 and 129
which we interpret as the Plan 9 file server and auth server. Replies to these options contain a
list of IP addresses for possible file servers and auth servers.
h the hostname to add to DHCP requests. Some DHCP servers, such as the one used by Com-
cast, will not respond unless a correct hostname is in the request.
m the maximum IP packet size to use on this interface.
n determine parameters but don’t configure the interface.
N look in /lib/ndb for the IP parameters. This only works if the interface is an ethernet. It uses the ethernet address to find a matching entry.

O addresses specified on the command line override those obtained via DHCP. A command line address of 0 implies no override.

p write configuration information to /net/ndb, even if other network interfaces are already configured

P do not write configuration information to /net/ndb, even if this is the first network interface to be configured

r by default, ipconfig exits after trying DHCP for 15 seconds with no answer. This option directs ipconfig instead to fork a background process that keeps trying forever.

u disable IPv6 duplicate discovery detection, which removes any existing ARP table entry for one of our IPv6 addresses before adding new ones.

x use the IP stack mounted at netmtpt instead of at /net.

X don’t fork a process to keep the DHCP lease alive.

o adds dhcpoption to the list of parameters requested of the DHCP server. The result will appear in /net/ndb should this be the first interface. The known options are:

arptimeout, baddr, bflen, bootfile, clientid, cookie, discovermask, discoverrouter, dns, dom, dumpfile, ethereencap, extpath, finger, homeagent, impress, ipaddr, ipforward, ipgw, ipmask, irc, lease, log, lpr, maxdatagram, maxmsg, message, mtu, name, netbiosdss, netbiosns, netbiosscope, netbiotype, ni, nisdomain, nisplus, nisplusdomain, nntp, nonlocal, ntp, overload, params, pathplateau, pathtimeout, policyfilter, pop3, rebindingtime, renewaltime, rl, rootpath, rs, serverid, smtp, st, staticroutes, stdar, subnetslocal, supplymask, swap, sys, tcpka, tcpkag, tcpp_tl, tftp, time, timeout, trailerencap, ttl, type, vendorclass, www, xdispmanager, xfont

The options ipmask, ipgw, dns, sys, and ntp are always requested.

If DHCP is requested, a process is forked off to renew the lease before it runs out. If the lease does run out, this process will remove any configured addresses from the interface.

Rip runs the routing protocol RIP. It listens for RIP packets on connected networks and updates the kernel routing tables. The options are:

b broadcasts routing information onto the networks.

n gathers routing information but doesn’t write to the route table. This is useful with –d to debug a network.

x use the IP stack mounted at netmtpt instead of at /net.

d turn on (voluminous) debugging.

Linklocal prints the IPv6 link-local address corresponding to the given mac address. Given –t, linklocal instead prints the 6to4 EUI-64-based IPv6 address corresponding to mac and 6to4 gateway gwipv4.

Ipv6on uses the network database at ndbfile to configure the network mounted on netmtpt with a link-local address (derived from its MAC address) and attempts to add a default IPv6 route to the local IPv4 gateway’s IPv6 address. If gwipv4 is supplied, it will be used as the gateway IPv4 address.

EXAMPLES

Configure Ethernet 0 as the primary IP interface. Get all addresses via DHCP. Start up a connection server and DNS resolver for this IP stack.

% bind –b ’#10’ /net
% bind –a ’#10’ /net
% ip/ipconfig
% ndb/cs
% ndb/dns –r
Add a second address to the stack.

% ip/ipconfig ether /net/ether0 add 12.1.1.2 255.255.255.0

At Bell Labs, our primary IP stack is always to the company’s internal firewall-protected network. The following creates an external IP stack to directly access the outside Internet. Note that the connection server uses a different set of ndb files. This prevents us from confusing inside and outside name/address bindings.

% bind -b '1l' /net.alt
% bind -b '1l' /net.alt
% ip/ipconfig -x /net.alt -g 204.178.31.1 ether /net.alt/ether1 \ 204.178.31.6 255.255.255.0
% ndb/cs -x /net.alt -f /lib/ndb/external
% ndb/dns -sx /net.alt -f /lib/ndb/external
% aux/listen -d /rc/bin/service.alt /net.alt/tcp

Get all addresses via DHCP. Configure the IPv6 link-local address automatically and listen for router announcements.

ip/ipconfig -6
ip/ipconfig ra6 recvra 1

FILES
/sys/log/v6routeradv

SOURCE
/sys/src/cmd/ip/ipconfig
/sys/src/cmd/ip/rip.c
/sys/src/cmd/ip/linklocal.c
/rc/bin/ipv6on

SEE ALSO
ether(3), ip(3), loopback(3), ndb(6), 6in4(8), dhcpd(8), ppp(8)
/lib/rfc/rfc2373 for IPv6’s modified EUI-64
NAME
ipok – verify that an IP address belongs to a country approved of by the US government

SYNOPSIS
aux/ipok IP ...

DESCRIPTION
Ipok consults various databases, including making whois queries, to determine what country a
given IP address belongs to and compares that against a list of known bad guys. For each IP, ipok
will print a line of the form
okay address–range country address–range–name owner [other ...]
where okay will be ok or bad.

EXAMPLES
See if ehime is in a terrorist country.
% aux/ipok 135.104.9.17
ok 135.104.0.0–135.104.255.255 US LUCENT–135–104–0–0–B 'Bell Laboratories/Lucent Technologies ' (cached)

FILES
/sys/lib/ipok.cache pre-loaded entries

SEE ALSO
ipokfs(4)
US State Department’s list of countries designated as supporting terrorist activities, defined by the
Export Administration Act, section 6(j), listed for example in

BUGS
Only works for IPv4 addresses.
NAME
telnetd, rlogind, rexec, ftpd – Internet remote access daemons

SYNOPSIS
ip/telnetd [-adnpt] [-u user]
ip/rlogind
ip/rexexec
ip/ftpd [-aAde] [-n namespace-file]

DESCRIPTION
These programs support remote access across the Internet. All expect the network connection to be standard input, output, and error. They are normally started from scripts in /rc/bin/service (see listen(8)).

telnetd allows login from a remote client. There are three types of login:

normal Normal users log in by encrypting and returning a challenge printed by telnetd. The user can use either the netkey program (see passwd(1)) or a SecureNet handheld authenticator to encrypt the challenge. /lib/namespace defines the namespace.

noworld Users in group noworld in /adm/users authenticate with a password in the clear. /lib/namespace.noworld defines the namespace.

anonymous User none requires no authentication. /lib/namespace defines the namespace.

telnetd’s options are:
a allow anonymous login by none
d print debugging to standard error
p don’t originate any telnet control codes
n turn on local character echoing and imply the p option
t trusted, that is, don’t authenticate
u use user as the local account name
N permit connections by ‘noworld’ users only.

rlogind logs in using the BSD remote login protocol. rlogind execs telnetd –nu after completing its initial handshake.

rexexec executes a command locally for a remote client. It uses the standard Plan 9 authentication (see authsrv(6)).

ftpd runs the Internet file transfer protocol. Users may transfer files in either direction between the local and remote machines. As for telnetd, there are three types of login:

normal Normal users authenticate via the same challenge/response as for telnetd. /usr/username/lib/namespace.ftp or, if that file does not exist, /lib/namespace defines the namespace.

noworld Users in group noworld in /adm/users login using a password in the clear. /lib/namespace.noworld defines the namespace.

anonymous Users anonymous and none require no authentication. The argument to the -n option (default /lib/namespace.ftp) defines the namespace. Anonymous users may only store files in the subtree below /incoming.

ftpd’s options are:
a allow anonymous access
A allow only anonymous access
d write debugging output to standard error
e treat any user as anonymous
n the namespace for anonymous users (default /lib/namespace.ftp)

To preserve intended protections in shared file trees, any directory containing a file .httplogin is locked by ftpd; see httpd(8).
FILES
/lib/namespace
/usr/username/lib/namespace.ftp
/lib/namespace.world
/lib/namespace.ftp

SOURCE
/sys/src/cmd/ip/telnetd.c
/sys/src/cmd/ip/rlogind.c
/sys/src/cmd/ip/rexexec.c
/sys/src/cmd/ip/ftpd.c

SEE ALSO
ftpfs(4), pop3(8)
NAME
kfccmd, ksync – kfs administration

SYNOPSIS
disk/kfccmd [-n name] cmd ...
disk/ksync

DESCRIPTION
Kfs is a local user-level file server for a Plan 9 terminal with a disk. Kfccmd transmits commands
to the kfs server (see kfs(4)). The –n option changes the name of the kfs service to kFs. name (by
default, full name is just kFs).
Ksync executes the sync command for all active kFs servers.
The known commands are described below. Note that some commands are multiple words and
should be quoted to appear as a single argument to rc(1).
allow Turn permission checking off (to simplify administration).
allowoff
disallow Turn permission checking on.
noauth Disable authentication of users.
halt Write all changed blocks and stop the file system.
start The opposite of halt; restart the file system.
help Print the list of commands.
rename file name
    Change the name of file to name. Name may be a single path element or a full path;
    if it is a full path, every element along the path must exist except the last.
newuser user
    Add user to /adm/users and make the standard directories needed for booting.
remove file
    Remove file and place its blocks on the free list.
crl file
    Remove file but do not place the blocks on the free list. This command can be used
to remove files that have duplicated blocks. The non–duplicate blocks can be
retrieved by checking the file system with option £ (see below).
create file owner group mode [adl]
    Create the file. Owner and group are users in /adm/users and mode is an octal
    number. If present, a creates an append only file, d creates a directory, and l cre-
    ates a file that is exclusive–use.
sync Write to disk all of the dirty blocks in the memory cache.
atime Toggle whether atimes are updated as files and directories are accessed. By default,
atimes are updated. On laptops it can be useful to turn off atime updates to reduce
disk accesses.
stats Report statistics about the performance of the file system.
user Re–initialize authentication information by reading /adm/users.
nowritegroup
    Each time kfs rereads /adm/users, it looks for a group named write. If such a
group exists, then the entire file system will appear read–only to users not in the
group. If a write group exists but no one is in it, it will be impossible to edit
/adm/users to correct the problem. To resolve this, the nowritegroup com-
mmand turns off write group checking until the next time /adm/users is reread.
cfs filsys
    Change the ‘console’ to the named file system (default is the main system).
chat Toggle tracing of 9P messages.
check [cdfPqrtw]
    Check the file system and print summary information. The options are
c  fix bad tags and clear the contents of the block.
d  delete redundant references to a block, fix bad UTF filenames.
f  rebuild the list of free blocks.
p  print the names of directories as they are checked.
P  print the names of all files as they are checked.
q  quiet mode: report errors, but suppress summary information
r  read all of the data blocks and check the tags.
t  fix bad tags.
w  write all of the blocks that are touched.

listen [address]
Start a listener to serve the network at address, default tcp!*!564. This feature
is intended to facilitate small networks of a couple machines in the situation when
convenience is more important than performance. This command is only useful on
machines with (possibly simulated) NVRAM, which needs to be readable to the kfs
processes; see readnvr in authsrv(2). The production file server (see fs(4)) is
strongly encouraged for anything more than casual use.

noneattach
When listening to the network, the default behavior is that the user none may only
attach over connections that have already authenticated as someone else. This pre-
vents just anyone from being able to dial your server and attach as none. The
noneattach command toggles whether none can attach without such a chaper-
one.

SOURCE
/sys/src/cmd/disk/kfscmd.c
/$objtype/bin/disk/ksync

SEE ALSO
  kfs(4), mkfs(8), prep(8), sd(3)
NAME

listen, listen1, tcp7, tcp9, tcp19, tcp21, tcp22, tcp23, tcp25, tcp53, tcp110, tcp113, tcp143, tcp513, tcp515, tcp564, tcp565, tcp566, tcp567, tcp993, tcp995, tcp1723, tcp17007, tcp17008, tcp17009, tcp17010, tcp17013 - listen for calls on a network device

SYNOPSIS

aux/listen [-iq] [-d srvdir] [-t trustsrvdir] [-n namespace] [net]  
aux/listen1 [-tv] addr cmd [args...]

DESCRIPTION

*listen* listens on a network for inbound calls to local services. *Net* is the network protocol on which to listen, by default /net/tcp. The services available are executable, non-empty files in *srvdir* or *trustsrvdir*. If neither *srvdir* nor *trustsrvdir* is given, *listen* looks for executable files in /bin/service. Services found in *srvdir* are executed as user none; services found in *trustsrvdir* are executed as the user who started *listen*. When changing user to none, a new namespace is created, usually by executing /lib/namespace, but -n selects an alternate *namespace*. Option -q suppresses affirmative log information. Option -i suppresses the periodic scan of the service directories for changes.

Service names are made by concatenating the name of the network with the name of the service or port. For example, an inbound call on the TCP network for port 565 executes service tcp565.

At least the following services are available in /bin/service.

tcp564 serve a piece of the name space using the Plan 9 file system protocol, with authentication via Tauth (in attach(5)), no encryption, and multiplex multiple users on a single connection (used by cpu(4), and also by Unix systems to see Plan 9 files).

tcp17007 serve a piece of the name space using the Plan 9 file system protocol, with authentication at the start, optional SSL encryption, and no multiplexing of users (typically used by cpu(1) and import(4)). Not usable by user none.

tcp17008 like tcp17007, but serves the root of the tree, forgoing the negotiation for which subtree to serve.

tcp17009 rx remote execution.

tcp17010 server for cpu(1) command.

tcp17013 server for old cpu(1) command for compatibility with old clients.

tcp7 echo any bytes received (bit mirror)

tcp9 consume any bytes received (bit bucket)

tcp19 chargen service.

tcp21 FTP daemon

tcp22 ssh(1) 'secure shell' encrypted terminal connection or file transfer.

tcp23 telnet terminal connection.

tcp25 mail delivery.

tcp53 TCP port for DNS.

tcp110 POP3 port.

tcp113 Identi port (always reports none).

tcp143 IMAP4rev1 port.

tcp513 rlogin terminal connection.

tcp515 LP daemon; see lp(8).

tcp565 report the address of the incoming call.

tcp993 Secure IMAP4rev1 port.

tcp995 Secure POP3 port.

tcp1723 PPTP (point-to-point tunnelling protocol) service.

At least the following services are available in /bin/service.auth, the usual *trustsrvdir*.

tcp566 validate a SecureNet box.

tcp567 Plan 9 authentication–ticket service.

*Listen1* is a lightweight listener intended for personal use, modeled from Inferno’s *listen(1)*. announces on *address*, running *cmd args*... for each incoming connection; the network directory is passed in the environment as $net. Option -t causes *listen1* to run as the invoking user; the default is to become none before listening. Option -v causes verbose logging on standard
output. See /rc/bin/tlssrvtunnel for an example.

FILES
/net/tcp by convention, TCP device bind point

SOURCE
/sys/src/cmd/aux/listen*.c
/rc/bin/service*

SEE ALSO
authsrv(6), dial(2)

BUGS
Srvdir, trustsvrdir and namespace must all be absolute path names.
NAME
lp – PostScript preprocessors

DESCRIPTION
These programs are part of the lp(1) suite. Each corresponds to a process in the –process option of lp and exists as an rc(1) script in /sys/lib/lp/process that provides an interface to a PostScript conversion program in /$cputype/bin/aux. The list of processors follows; after each description is a bracketed list of lp options to which the processor responds:
dpost converts troff(1) output for device post to PostScript. This is used for files troff’ed on
our UNIX systems that do not handle UTF characters. [DLcimnorxy]
dvipost converts tex(1) output to PostScript. [Lcinor]
g3post converts CCITT Group 3 FAX data to PostScript. [DLm]
gifpost converts GIF image data to PostScript. [DLm]
generic is the default processor. It uses file(1) to determine the type of input and executes the
correct processor for a given (input, printer) pair.
hpost adds a header page to the beginning of a PostScript printer job so that it may be sepa-
rated from other jobs in the output bin. The header has the image of the job’s owner from
the directory of faces (see face(6)). Page reversal is also done in this processor.
jpgpost converts JPEG image data to PostScript. [DLm]
noproc passes files through untouched.
p9bitpost converts a Plan 9 image to PostScript, such as /dev/screen for the whole screen,
/dev/window for that window’s data, and /dev/wsys/.../window for some
other window’s data. [DLm]
pdfpost converts PDF data to PostScript.
post passes PostScript through, adding option patches for paper tray information. This
does not always work with PostScript generated on other systems.
ppassport converts UTF text to PostScript. [DLcifilmnorxy]
tr2post converts troff(1) output for device utf (the default) to PostScript. See
/sys/lib/troff/font/devutf directory for troff font width table descriptions.
See also the /sys/lib/postscript/troff directory for mappings of troff UTF
character space to PostScript font space. [DLcimnorxy]

SOURCE
/sys/src/cmd/postscript

SEE ALSO
lp(1)

BUGS
The file command is not always smart enough to deal with certain file types. There are PostScript
conversion programs that do not have processors to drive them.
NAME
dump9660, mk9660 - create an ISO-9660 CD image

SYNOPSIS
disk/mk9660 [-D][-9cj][-b bootfile][-B bootfile][-p proto][-s src][-v volume]
   image
disk/dump9660 [-D][-9cj][-p proto][-s src][-v volume][-m maxsize][-n now]
   image

DESCRIPTION
Mk9660 writes to the random access file image an ISO-9660 CD image containing the files named
in proto (by default, /sys/lib/sysconfig/proto/portproto) from the file tree src (by
default, the current directory). The proto file is formatted as described in mkfs(8).
The created CD image will be in ISO-9660 format, but by default the file names will be stored in
UTF-8 with no imposed length or character restrictions. The -c flag causes mk9660 to use only
file names in "8.3" form that use digits, letters, and underscore. File names that do not conform
are changed to Dnnnnnn (for directories) or Fnnnnnn (for files); a key file _CONFORM.MAP is cre-
ated in the root directory to ease the reverse process.
If the -9 flag is given, the system use fields at the end of each directory entry will be populated
with Plan directory information (owner, group, mode, full name); this is interpreted by 9660srv.
If the -j flag is given, the usual directory tree is written, but an additional tree in Microsoft Joliet
format is also added. This second tree can contain long Unicode file names, and can be read by
9660srv as well as most versions of Windows and many Unix clones. The characters *, :, ;, ?,
and \ are allowed in Plan 9 file names but not in Joliet file names; non-conforming file names are
translated and a _CONFORM.MAP file written as in the case of the -c option.
If the -x flag is given, Rock Ridge extensions are written in the format of the system use sharing
protocol; this format provides Posix-style file metadata and is common on Unix platforms.
The options -c, -9, -j, and -x may be mixed freely with the exception that -9 and -x are
mutually exclusive.
The -v flag sets the volume title; if unspecified, the base name of proto is used.
The -: flag causes mk9660 to replace colons in scanned file names with spaces; this is the
inverse of the map applied by dossrv(4) and is useful for writing Joliet CDs containing data from
FAT file systems.
The -b option creates a bootable CD. Bootable CDs contain pointers to floppy images which are
loaded and booted by the BIOS. Bootfile should be the name of the floppy image to use; it is a
path relative to the root of the created CD. That is, the boot floppy image must be listed in the
proto file already: the -b option just creates a pointer to it.
The -B option is similar to -b but the created CD image is marked as having a non-floppy-
emulation boot block. This gives the program in the boot block full (ATA) LBA access to the CD
filesystem, not just the initial blocks that would fit on a floppy.
The -D flag creates immense amounts of debugging output on standard error.
Dump9660 is similar in specification to mk9660 but creates and updates backup CD images in the
style of the dump file system (see fs(4)). The dump is file-based rather than block-based: if a file's
contents have not changed since the last backup, only its directory entry will be rewritten.
The -n option specifies a time (in seconds since January 1, 1970) to be used for naming the dump
directory.
The -m option specifies a maximum size for the image; if a backup would cause the image to grow
larger than maxsize, it will not be written, and dump9660 will exit with a non-empty status.

EXAMPLE
Create an image of the Plan 9 source tree, including a conformant ISO-9660 directory tree, Plan 9
extensions in the system use fields, and a Joliet directory tree.
   disk/mk9660 -9cj -s /sys/src \n   -p /sys/lib/sysconfig/proto/allproto cdimage
SOURCE
/sys/src/cmd/disk/9660

SEE ALSO
9660srv (in dossrv(4)), cdfs(4), mkfs(8)
NAME
mkflashfs – make a journalling file system for flash memory

SYNOPSIS
aux/mkflashfs [−n nsect][−z sectsize] file

DESCRIPTION
Mkflashfs creates an empty journalling file system in file, typically /dev/flash/flash.
The files and directory structure are divided into sectsize (default 4096) byte blocks. Larger
blocks make large files more compact but take longer to access. Supplying the −n option forces
file to contain exactly nsect sectors.

SOURCE
/sys/src/cmd/aux/flashfs/mkflashfs.c

SEE ALSO
flashfs(4), paqfs(4), sacfs(4)
NAME
mkfs, mkext - archive or update a file system

SYNOPSIS
disk/mkfs [-aprvxU] [-d root] [-n name] [-s source] [-u users] [-z n] proto ...
disk/mkext [-d name] [-u] [-h] [-v] [-x] [-T] file ...

DESCRIPTION
Mkfs copies files from the file tree source (default /) to a kfs file system (see kfs(4)). The kfs service is mounted on root (default /n/kfs), and /adm/users is copied to root/adm/users. The proto files are read (see proto(2)) for their format and any files specified in them that are out of date are copied to /n/kfs.

Mkfs copies only those files that are out of date. Such a file is first copied into a temporary file in the appropriate destination directory and then moved to the destination file. Files in the kfs file system that are not specified in the proto file are not updated and not removed.

The options to mkfs are:
a  Instead of writing to a kfs file system, write an archive file to standard output, suitable for mkext. All files in proto, not just those out of date, are archived.
 x  For use with -a, this option writes a list of file names, dates, and sizes to standard output rather than producing an archive file.
d  root  Copy files into the tree rooted at root (default /n/kfs). This option suppresses setting the uid and gid fields when copying files. Use -U to reenable it.
n  name  Use kfs.name as the name of the kfs service (default kfs).
p  Update the permissions of a file even if it is up to date.
r  Copy all files.
s  source  Copy from files rooted at the tree source.
u  users  Copy file users into /adm/users in the new system.
v  Print the names of all of the files as they are copied.
z  n  Copy files assuming kfs block n (default 1024) bytes long. If a block contains only 0-valued bytes, it is not copied.

Mkext unpacks archive files made by the -a option of mkfs. Each file on the command line is unpacked in one pass through the archive. If the file is a directory, all files and subdirectories of that directory are also unpacked. When a file is unpacked, the entire path is created if it does not exist. If no files are specified, the entire archive is unpacked; in this case, missing intermediate directories are not created. The options are:
d  specifies a directory (default /) to serve as the root of the unpacked file system.
u  sets the owners of the files created to correspond to those in the archive and restores the modification times of the files.
T  restores only the modification times of the files.
v  prints the names and sizes of files as they are extracted.
h  prints headers for the files on standard output instead of unpacking the files.

EXAMPLES
Make an archive to establish a new file system:
disk/mkfs -a -u files/adm.users -s dist proto > arch

Unpack that archive onto a new file system:
srv newfs
mount -c /srv/newfs /n/newfs
disk/mkext -u -d /n/newfs < arch

SOURCE
/sys/src/cmd/disk/mkfs.c
/sys/src/cmd/disk/mkext.c

SEE ALSO
prep(8), kfscmd(8), s(3), tar(1)
NAME
    mkpaqfs - make a compressed read-only file system

SYNOPSIS
    mkpaqfs [-u][ -b blocksize][ -l label][ -o file][ source]

DESCRIPTION
    Mkpaqfs copies files from the file tree source (default .) to the paqfs(4) file system archive file.
    The files and directory structure are divided into blocksize (default 4096) byte blocks. Larger blocks make larger files more compact but take longer to access. Blocksize must be in the range of 512 bytes to 52K bytes. If the -u option is set, the blocks are not compressed. Otherwise each block is compressed using the flate(2) compression algorithm. The -l option embeds a label of up to 32 bytes within the file header and may be useful for identifying the file system.

SOURCE
    /sys/src/cmd/paqfs/mkpaqfs.c

SEE ALSO
    paqfs(4)
NAME
   mksacfs - make a compressed file system

SYNOPSIS
   disk/mksacfs [-u] [-b blocksize] [-o file] source

DESCRIPTION
   Mksacfs copies files from the file tree source (default .) to a the sacfs(4) file system archive file.
   The files and directory structure are divided into blocksize (default 4096) byte blocks. Larger blocks make large files more compact but take longer to access. Blocksize must be at least 116. If -u is given, the blocks are not compressed. Otherwise each block is compressed using an efficient compression algorithm.

SOURCE
   /sys/src/cmd/disk/sacfs/mksacfs.c

SEE ALSO
   sacfs(4)
NAME
aux/mouse, aux/accupoint - configure a mouse to a port

SYNOPSIS
aux/mouse [-b baud] [-d type] [-n] port
aux/accupoint

DESCRIPTION
Mouse queries a mouse on a serial or PS2 port for its type and then configures the port and the
mouse to be used to control the cursor.

Port can be either a port number (e.g. 0 or 1) or the string ps2 or ps2intellimouse. The initial-
zation can be automated by setting mouseport in plan9.ini(8), which will enable a call to
mouse in termrc (see cpurc(8)).

The option -d provides a default mouse type should mouse fail to determine it. The types are:
C Logitech type C mouse
W Logitech type W mouse
M Microsoft compatible mouse

The -n flag queries the mouse and reports its type but does not set the device type.
The -b flag sets the baud rate for communication; it is effectual only for serial mice.

Accupoint is a process, to be used with pipefile(1), that processes events from an AccuPoint II
pointing device with four buttons, such as on Toshiba Portégé 3440CT and 3480CT laptops, con-
verting events on the two extra buttons (which appear as buttons 4 and 5 in the mouse(3) inter-
face) into a simulation of button 2. These extra buttons on laptops are in turn simulations of Intell-
limouse scrolling buttons and have peculiar properties: they generate only ‘down’ events that
repeat automatically, like a keypad, in an approximation of the Intellimouse scroll wheel. Accupoint
overcomes this behavior to produce a reasonable approximation of a normal mouse but-
tton 2: it makes left button act like a regular button 2, but is slow to release (the program must
wait for a repeat time before it knows the button has been released), while the right button gener-
ates a fast button 2 ‘click’. To use accupoint, add a line like this to
/usr/$user/lib/profile or to a system-dependent configuration script in termrc (see cpurc(8)):

    pipefile -dr /bin/aux/accupoint /dev/mouse

Before running accupoint, the mouse should be configured as an intellimouse or
ps2intellimouse.

SOURCE
/sys/src/cmd/aux/mouse.c
/sys/src/cmd/aux/accupoint.c

SEE ALSO
cons(3), cpurc(8), pipefile(1).

BUGS
Due to the limitations of pipefile(1), when running accupoint it is difficult restart rio(1) if it has
exited.
NAME
    na - assembler for the Symbios Logic PCI–SCSI I/O Processors

SYNOPSIS
    aux/na file

DESCRIPTION
    The SYM53C8XX series of PCI–SCSI I/O Processors contain loadable microcode to control their
    operation. The microcode is written in a language called SCRIPTS. Aux/na is an assembler for the
    SCRIPTS programming language. It assembles SCRIPTS code in file into an array of assembled
    SCRIPTS instructions, patches, defines and enums that can be included in a C device driver.

SOURCE
    /sys/src/cmd/aux/na

SEE ALSO
    Symbios Logic, "PCI–SCSI I/O Processors Programming Guide Version 2.1"
    /sys/src/9/*/sd53c8xx.n   SCRIPTS source code
    /sys/src/9/*/sd53c8xx.c   driver for the SYM53C8XX series of PCI–SCSI controllers

AUTHOR
    Nigel Roles (ngr@9fs.org)
NAME
query, ipquery, mkhash, mkdb, mkhosts, cs, csquery, dns, dnstcp, dnsquery, dnsdebug, inform - network database

SYNOPSIS
ndb/query [-am] [-f dbfile] attr value [ rattr ]
ndb/ipquery attr value rattr...
ndb/mkhash file attr
ndb/mkdb
ndb/mkhosts [ domain [ dbfile ]]
ndb/cs [-n] [-f dbfile] [-x netmtpt]
ndb/csquery [-s] [ server [ addr... ]] 
ndb/dnstcp [-rR] [-f dbfile ] [-x netmtpt ] [ conn-dir ]
ndb/dnsquery
ndb/dnsdebug [-rx] [-f dbfile ] [[@server] domain-name [ type ]]
ndb/inform [-x netmtpt ]

DESCRIPTION
The network database holds administrative information used by network programs such as
dhcpd(8), ipconfig(8), con(1), etc.

Ndb/query searches the database dbfile (/lib/ndb/local by default) for an attribute of type
attr and value value. If rattr is not specified, all entries matched by the search are printed. If rattr
is specified, the value of the first pair with attribute rattr of all the matched entries normally is
printed. Under -m and rattr, the values of all pairs with a rattr attribute within the first matching
entry are printed. Under -a and rattr, all values of pairs with a rattr attribute within all entries
are printed.

Ndb/ipquery uses ndbipinfo (see ndb(2)) to search for the values of the attributes rattr corre-
sponding to the system with entries of attribute type attr and value value.

Ndb/inform sends an RFC2136 DNS inform packet to a nameserver to associate the host’s IPv4
address with its DNS name. This is required if the domain’s nameserver is a Microsoft Windows
Active Directory controller.

Database maintenance
Ndb/mkhash creates a hash file for all entries with attribute attr in database file file. The hash files
are used by ndb/query and by the ndb library routines.

Ndb/mkdb is used in concert with awk(1) scripts to convert uucp systems files and IP host files
into database files. It is very specific to the situation at Murray Hill.

When the database files change underfoot, ndb/cs and ndb/dns track them properly. Nonetheless,
to keep the database searches efficient it is necessary to run ndb/mkhash whenever the files are
modified. It may be profitable to control this by a frequent cron(8) job.

Ndb/mkhosts generates a BSD style hosts, hosts.txt, and hosts.equiv files from an ndb
data base file specified on the command line (default /lib/ndb/local). For local reasons the
files are called hosts.1127, astro.txt, and hosts.equiv.

Connection service
Ndb/cs is a server used by dial(2) to translate network names. It is started at boot time. It finds
out what networks are configured by looking for /net/*/clone when it starts. It can also be
told about networks by writing to /net/cs a message of the form:
        add net1 net2 ...

Ndb/cs also sets the system name in /dev/sysname if it can figure it out. The options are:
-f  supplies the name of the data base file to use, default /lib/ndb/local.
-n  causes cs to do nothing but set the system name.
-x  specifies the mount point of the network.

Ndb/csquery queries ndb/cs to see how it resolves addresses. Ndb/csquery prompts for addresses
and prints what ndb/cs returns. Server defaults to /net/cs. If any addr are specified,
ndb/csquery prints their translations and immediately exits. The exit status will be nil only if all addresses were successfully translated. The -s flag sets exit status without printing any results.

Domain name service

Ndbs serves ndb/cs and remote systems by translating Internet domain names. Ndb/dns is started at boot time. By default dns serves only requests written to /net/dns. Programs must seek to offset 0 before reading or writing /net/dns or /net/cs. The options are:

- a sets the maximum time in seconds that an unreferenced domain name will remain cached. The default is one hour (3600).
- f supplies the name of the data base file to use, default /lib/ndb/local.
- n whenever a DNS zone that we serve changes, send UDP NOTIFY messages to any dns slaves for that zone (see the dnsslave attribute below).
- N sets the goal for the number of domain names cached to target rather than the default of 8,000.
- o used with -s, -o causes dns to assume that it straddles inside and outside networks and that the outside network is mounted on /net.alt. Queries for inside addresses will be sent via /net/udp (or /net/tcp in response to truncated replies) and those for outside addresses via /net.alt/udp (or /net.alt/tcp). This makes dns suitable for serving non-Plan-9 systems in an organization with firewalls, DNS proxies, etc., particularly if they don't work very well. See 'Straddling Server' below for details.
- r act as a resolver only: send 'recursive' queries, asking the other servers to complete lookups. If present, /env/DNSSERVER must be a space-separated list of such DNS servers' IP addresses, otherwise optional ndb(6) dns attributes name DNS servers to forward queries to.
- R ignore the 'recursive' bit on incoming requests. Do not complete lookups on behalf of remote systems.
- s also answer domain requests sent to UDP port 53.
- x specifies the mount point of the network.
- z whenever we receive a UDP NOTIFY message, run program with the domain name of the area as its argument.

When the -r option is specified, the servers used come from the dns attribute in the database. For example, to specify a set of dns servers that will resolve requests for systems on the network mh-net:

```plaintext
ipnet=mh-net ip=135.104.0.0 ipmask=255.255.0.0
dns=ns1.cs.bell-labs.com
dns=ns2.cs.bell-labs.com
dom=ns1.cs.bell-labs.com ip=135.104.1.11
dom=ns2.cs.bell-labs.com ip=135.104.1.12
```

The server for a domain is indicated by a database entry containing both a dom and a ns attribute.

```plaintext
dom=
  ns=A.ROOT-SERVERS.NET
  ns=B.ROOT-SERVERS.NET
  ns=C.ROOT-SERVERS.NET
dom=A.ROOT-SERVERS.NET ip=198.41.0.4
dom=B.ROOT-SERVERS.NET ip=128.9.0.107
dom=C.ROOT-SERVERS.NET ip=192.33.4.12
```

The last three lines provide a mapping for the server names to their ip addresses. This is only a hint and will be superseded from whatever is learned from servers owning the domain.

Authoritative Name Servers

You can also serve a subtree of the domain name space from the local database. You indicate subtrees that you would like to serve by adding an soa= attribute to the root entry. For example, the Bell Labs CS research domain is:

```plaintext
dom=cs.bell-labs.com soa=
  refresh=3600 ttl=3600
  ns=plan9.bell-labs.com
  ns=ns1.cs.bell-labs.com
  ns=ns2.cs.bell-labs.com
```

894
mb=presotto@plan9.bell-labs.com
mx@mail.research.bell-labs.com pref=20
mx=plan9.bell-labs.com pref=10
dnsslave=nslocum.cs.bell-labs.com
dnsslave=vex.cs.bell-labs.com
mx=mail.research.bell-labs.com pref=20
mb=presotto@plan9.bell-labs.com
mx@mail.research.bell-labs.com pref=20
mx=plan9.bell-labs.com pref=10
dnsslave=nslocum.cs.bell-labs.com
dnsslave=vex.cs.bell-labs.com
mx=mail.research.bell-labs.com pref=20

Here, the mb entry is the mail address of the person responsible for the domain (default postmaster). The mx entries list mail exchangers for the domain name and refresh and ttl define the area refresh interval and the minimum TTL for records in this domain. The dnsslave entries specify slave DNS servers that should be notified when the domain changes. The notification also requires the --n flag.

Reverse Domains
You can also serve reverse lookups (returning the name that goes with an IP address) by adding an soa= attribute to the entry defining the root of the reverse space.

For example, to provide reverse lookup for all addresses in starting with 135.104 or fd00::, ndb must contain a record like:

```
dom=104.135.in-addr.arpa soa=
dom=d.f.ip6.arpa soa= # special case, rfc 4193
refresh=3600 ttl=3600
ns=plan9.bell-labs.com
ns=ns1.cs.bell-labs.com
ns=ns2.cs.bell-labs.com
```

Notice the form of the reverse address. For IPv4, it's the bytes of the address range you are serving reversed and expressed in decimal, and with .in-addr.arpa appended. For IPv6, it's the nibbles (4-bit fields) of the address range you are serving reversed and expressed in hexadecimal, and with .ip6.arpa appended. These are the standard forms for a domain name in a PTR record.

If such an soa entry exists in the database, reverse addresses will automatically be generated from any IP addresses in the database that are under this root. For example

```
dom=ns1.cs.bell-labs.com ip=135.104.1.11
```

will automatically create both forward and reverse entries for ns1.cs.bell-labs.com. Unlike other DNS servers, there's no way to generate inconsistent forward and reverse entries.

Classless reverse delegation
Following RFC 2317, it is possible to serve reverse DNS data for IPv4 subnets smaller than /24. Declare the non--/24 subnet, the reverse domain and the individual systems.

For example, this is how to serve RFC-2317 ptr records for the subnet 65.14.39.128/123.

```
inet=our-t1 ip=65.14.39.128 ipmask=/123
dom=128.39.14.65.in-addr.arpa soa=
refresh=3600 ttl=3600
ns=ns1.our-domain.com
ns=ns2.our-domain.com
ip=65.14.39.129 dom=router.our-domain.com
```

Delegating Name Service Authority
Delegation of a further subtree to another set of name servers is indicated by an soa=delegated attribute.

```
dom=bignose.cs.research.bell-labs.com
soa=delegated
ns=anna.cs.research.bell-labs.com
ns=dj.cs.research.bell-labs.com
```

Nameservers within the delegated domain (as in this example) must have their IP addresses listed elsewhere in ndb files.

Wildcards, MX and CNAME records
Wild–carded domain names can also be used. For example, to specify a mail forwarder for all Bell Labs research systems:
domain=*\research.bell-labs.com
mx=\research.bell-labs.com

‘Cname’ aliases may be established by adding a cname attribute giving the real domain name; the
name attached to the dom attribute is the alias. ‘Cname’ aliases are severely restricted; the aliases
may have no other attributes than dom and are daily further restricted in their use by new RFCs.

cname=anna.cs.bell-labs.com dom=www.cs.bell-labs.com

makes www... a synonym for the canonical name anna... .

Straddling Server

Many companies have an inside network protected from outside access with firewalls. They usually
provide internal ‘root’ DNS servers (of varying reliability and correctness) that serve internal
domains and pass on DNS queries for outside domains to the outside, relaying the results back
and caching them for future use. Some companies don’t even let DNS queries nor replies through
their firewalls at all, in either direction.

In such a situation, running dns –so on a machine that imports access to the outside network via
/net.alt from a machine that straddles the firewalls, or that straddles the firewalls itself, will let
internal machines query such a machine and receive answers from outside nameservers for outside
addresses and inside nameservers for inside addresses, giving the appearance of a unified domain
name space, while bypassing the corporate DNS proxies or firewalls. This is different from running
dns –s and dns –sR /net.alt –f /lib/ndb/external on the same machine,
which keeps the inside and outside namespaces entirely separate.

Under –o, several sys names are significant: inside-dom, inside-ns, and outside-ns. Inside-dom should contain a series of dom pairs naming domains internal to the organization. Inside-ns should contain a series of ip pairs naming the internal DNS ‘root’ servers. Outside-ns should contain a series of ip pairs naming the external DNS servers to consult.

Zone Transfers and TCP

Dnstcp is invoked, usually from /rc/bin/service/tcp53, to answer DNS queries with long
answers via TCP, notably to transfer a zone within the database dbfile (default
/lib/ndb/local) to its invoker on the network at netmtpt (default /net). Standard input will
be read for DNS requests and the DNS answers will appear on standard output. Recursion is dis-
abled by –R; acting as a pure resolver is enabled by –r. If conn-dir is provided, it is assumed to be
a directory within netmtpt/tcp and is used to find the caller’s address.

DNS Queries and Debugging

Ndb/dnsquery can be used to query nbd/dns to see how it resolves requests. Ndb/dnsquery
prompts for commands of the form

domain-name request-type

where request-type can be ip, ipv6, mx, ns, cname, ptr.... In the case of the inverse query
type, ptr, dnsquery will reverse the ip address and tack on the .in–addr.arpa if necessary.

Ndb/dnsdebug is like nbd/dnsquery but bypasses the local server. It communicates via UDP (and
sometimes TCP) with the domain name servers in the same way that the local resolver would and
displays all packets received. The query can be specified on the command line or can be
prompted for. The queries look like those of nbd/dnsquery with one addition. Ndb/dnsdebug can
be directed to query a particular name server by the command @name–server. From that point on,
all queries go to that name server rather than being resolved by dnsdebug. The @ command
returns query resolution to dnsdebug. Finally, any command preceded by a @name–server sets the
name server only for that command.

Normally dnsdebug uses the /net interface and the database file /lib/ndb/local. The –f
option supplies the name of the data base file to use. The –r option is the same as for nbd/dns.
The –x option directs dnsdebug to use the /net.alt interface and /lib/ndb/external
database file.

EXAMPLES

Look up helix in nbd.

% nbd/query sys helix
sys=helix dom=helix.research.bell-labs.com bootf=/mips/9powerboot
ip=135.104.117.31 ether=080069020427

896
Look up plan9.bell-labs.com and its IP address in the DNS.

```bash
% ndb/dnsquery
> plan9.bell-labs.com ip
plan9.bell-labs.com ip 204.178.31.2
> 204.178.31.2 ptr
2.31.178.204.in-addr.arpa ptr plan9.bell-labs.com
> 2.31.178.204.in-addr.arpa ptr ampl.com
```

Print the names of all systems that boot via PXE.

```bash
% ndb/query -a bootf /386/9pxeload sys
```

**FILES**

- `/env/DNSSERVER` resolver's DNS servers' IP addresses.
- `/lib/ndb/local` first database file searched
- `/lib/ndb/local.*` hash files for `/lib/ndb/local`
- `/srv/cs` service file for `ndb/cs`
- `/net/cs` where `/srv/cs` gets mounted
- `/srv/dns` service file for `ndb/dns`
- `/net/dns` where `/srv/dns` gets mounted

**SOURCE**

`/sys/src/cmd/ndb`

**SEE ALSO**

`ndb(2), ndb(6)`

**BUGS**

*Nd* databases are case-sensitive: ethernet addresses must be in lower-case hexadecimal.
NAME
newuser - adding a new user

SYNOPSIS
/sys/lib/newuser

DESCRIPTION
To establish a new user on Plan 9, add the user to /adm/users by running the `uname` command
on the console of the file server (see `users(6)` and `fossilcons(8)`). Next, give the user a password
using `auth/changeuser` on the console of the authentication server (see `auth(8)`). At this point, the
user can bootstrap a terminal using the new name and password. The terminal will only get as far
as running `rc`, however, as no profile exists for the user.

The `rc(1)` script `/sys/lib/newuser` sets up a sensible environment for a new user of Plan 9.
Once the terminal is running `rc`, type

```
/sys/lib/newuser
```

to build the necessary directories in `/usr/$user`, create `/mail/box/$user/mbox`,
/cron/$user/cron, a reasonable initial profile in `/usr/$user/lib/profile` and plumbing
rules in `/usr/$user/lib/plumbing` (see `plumber(4)`). The script then runs the profile
which, as its last step, brings up `rio(1)`. At this point the user’s environment is established and
running. (There is no need to reboot.) It may be prudent at this point to run `passwd(1)` to change
the password, depending on how the initial password was chosen.

The profile built by `/sys/lib/newuser` looks like this:

```
bind -a $home/bin/rc /bin
bind -a $home/bin/$cputype /bin
bind -c tmp /tmp
font = /lib/font/bit/pelm/euro.9.font
switch($service){
  case terminal
    plumber
    upas/fs
    echo -n accelerated > '/m/mousectl'
    echo -n 'res 3' > '/m/mousectl'
    prompt=('term% ' ' ')
    fn term%{ $* }
    exec rio
  case cpu
    if (! test -e /mnt/term/mnt/wsys) {
      # cpu call from drawterm
      font=/lib/font/bit/pelm/latin1.8.font
      auth/factotum
      plumber
      exec rio
    }
    bind /mnt/term/dev/cons /dev/cons
    bind /mnt/term/dev/consctl /dev/consctl
    bind -a /mnt/term/dev /dev
    prompt=('cpu% ' ' ')
    fn cpu%{ $* }
    upas/fs
    news
  case con
```
prompt=('cpu% ' ' ')
news
}

Sites may make changes to /sys/lib/newuser that reflect the properties of the local environment.

SEE ALSO
passwd(1), rio(1), namespace(4), fossil(4), users(6), auth(8), fossilcons(8).
NAME
nfs, portmapper, pcnfsd – NFS service

SYNOPSIS
  aux/nfs [ rpc–options... ][ nfs–options... ]
  aux/pcnfsd [ rpc–options... ]
  aux/portmapper [ rpc–options... ]

DESCRIPTION
These programs collectively provide NFS access to Plan 9 file servers. *Nfs*, *pcnfsd*, and *portmapper* run on a Plan 9 CPU server, and should be started in that order. All users on client machines have the access privileges of the Plan 9 user `none`. Currently only NFS version 2 is served.

The *rpc–options* are all intended for debugging:
  −r    Reject: answer all RPC requests by returning the `AUTH_TOO_WEAK` error.
  −v    Verbose: show all RPC calls and internal program state, including RPC messages. (In any case, the program creates a file `/srv/name.chat where *name* is that of the program; echoing 1 or 0 into this file sets or clears the −v flag dynamically.)
  −D    Debug: show all RPC messages (at a lower level than −v). This flag may be repeated to get more detail.
  −C    Turn off caching: do not answer RPC requests using the RPC reply cache.

The *nfs–options* are:
  −a addr Set up NFS service for the Plan 9 server at network address `addr`.
  −f file Set up NFS service for the Plan 9 server at `file` (typically an entry in `/srv`).
  −n Do not allow per–user authentication (default and mandatory).
  −c file File contains the uid/gid map configuration. It is read at startup and subsequently every hour (or if `c` is echoed into `/srv/nfs.chat`). Blank lines or lines beginning with `#` are ignored; lines beginning with `!` are executed as commands; otherwise lines contain four fields separated by white space: a regular expression (in the notation of `regexp(6)`) for a class of servers, a regular expression for a class of clients, a file of user id’s (in the format of a Unix password file), and a file of group id’s (same format).
  −s Expect a network connection on file descriptor 1 instead of listening for incoming calls.
  −t Listen for incoming TCP calls, rather than UDP calls.

NFS clients must be in the Plan 9 `/lib/ndb` database. The machine name is deduced from the IP address via `ndb/query`. The machine name specified in the NFS Unix credentials is completely ignored.

Pcnfsd is a toy program that authorizes PC–NFS clients. All clients are mapped to uid=1, gid=1 (daemon on most systems) regardless of name or password.

EXAMPLES
A simple `/lib/ndb/nfs` might contain:

```
!9fs tcp!ivy
.+ [^.]\+.cvrd\./hall\.edu /n/ivy/etc/passwd /n/ivy/etc/group
```

A typical entry in `/rc/bin/cpurc` might be:

```
aux/nfs [−a tcp!pie −a tcp!yoshimi −c /lib/ndb/nfs
aux/pcnfsd
aux/portmapper
```

Assuming the CPU server’s name is `eduardo`, the mount commands on the client would be:

```
/etc/mount −o soft,intr eduardo:pie /n/pie
/etc/mount −o soft,intr eduardo:yoshimi /n/yoshimi
```

Note that a single instance of *nfs* may provide access to several 9P servers.
FILES
   /lib/ndb/nfs  List of uid/gid maps.
   /sys/log/nfs  Log file.

SOURCE
   /sys/src/cmd/9nfs

BUGS
   It would be nice to provide authentication for users, but Unix systems provide too low a level of security to be trusted in a Plan 9 world.

SEE ALSO
   nfs(4)
NAME
partfs - serve file, with partitions

SYNOPSIS
disk/partfs [-Dr][−d diskname][−m mtpt][−s srvname] diskimage

DESCRIPTION
Partfs presents the file diskimage in the manner of sd(3) on mtpt/diskname (default /dev/sdXX). Changes made to the disk are written through to diskimage unless the −r option is given.

When setting disk geometry with the geometry control message, the arguments are sectors and sector size.

The −s option causes partfs to post its 9P service at /srv/service.

EXAMPLES
Partition a USB flash device:

    usb/disk
    disk/partfs /n/disk/0/data
    disk/mbr −m /386/mbr /dev/sdXX/data
    disk/fdisk −baw /dev/sdXX/data
    disk/prep /dev/sdXX/plan9

SOURCE
/sys/src/cmd/disk/partfs.c

SEE ALSO
sd(3), disksim(8), prep(8)
NAME
pci - print PCI bus configuration

SYNOPSIS
pci[ −bv ]

DESCRIPTION
Pci normally prints one line per device found on the local PCI bus described by #$/pci. The fields are bus . device . function , class, class code, vendor / device ids , IRQ (interrupt), followed by the configuration registers in pairs of index : address and size. The −b option suppresses output for PCI bridges. The −v option adds a second line per device, containing an English description obtained from /lib/pci.

FILES
/lib/pci

SOURCE
/rc/bin/pci

SEE ALSO
pnp(3)
NAME
   pcmcia – identify a PCMCIA card

SYNOPSIS
   aux/pcmcia [file]

DESCRIPTION
   Aux/pcmcia translates the PCMCIA information structure from file (default #y/pcm0attr) into a readable description and writes it to standard output.

FILES
   #y/pcm0attr   The attribute memory of the card in the PCMCIA slot.

SOURCE
   /sys/src/cmd/aux/pcmcia.c
NAME
pemdecode, pemencode – encode files in Privacy Enhanced Mail (PEM) format

SYNOPSIS
auth/pemdecode section [ file ]
auth/pemencode section [ file ]

DESCRIPTION
PEM is a textual encoding for binary data originally used by the Privacy Enhanced Mail program but now commonly used for other applications, notably TLS. PEM encodes data in base 64 (see encode(2)) between lines of the form:

------BEGIN SECTION------
------END SECTION------

where SECTION may be any string describing the encoded data. The most common use of PEM format on Plan 9 is for encoding X.509 certificates; see rsa(8).

Pemdecode extracts the named section and writes the decoded data to standard output.
Pemencode encodes its standard input, labels it as a section, and writes it to standard output.

EXAMPLES
Encode and decode a simple greeting:

% echo hello world |
auth/pemencode GREETING
------BEGIN GREETING------
aGVsbG8gd29ybGQK
------END GREETING------
% echo hello world |
auth/pemencode GREETING |
auth/pemdecode GREETING
hello world
%

SOURCE
/sys/src/cmd/auth

SEE ALSO
rsa(8)
NAME
ping, gping, traceroute, hogports – probe the Internet

SYNOPSIS
ip/ping [-6aflqr][ -i interval][ -n count][ -s size][ -w waittime] destination
ip/gping [-r][-l][-i interval] destination [ destination ...]
ip/traceroute [ -dn][-a n][ -h nbuck][ -t sttl] dest
ip/hogports [ mtpt/] proto ! address ! startport[ -endpoint]

DESCRIPTION
Ping sends ICMP echo request messages to a system. It can be used to determine the network delay and whether or not the destination is up. By default, a line is written to standard output for each request. If a reply is received the line contains the request id (starting at 0 and incrementing), the round trip time for this request, the average round trip time, and the time to live in the reply packet. If no reply is received the line contains the word "lost", the request id, and the average round trip time.

If a reply is received for each request, ping returns successfully. Otherwise it returns an error status of "lost messages".

The options are:
6 force the use of IPv6’s ICMP, icmpv6, instead of IPv4’s ICMP. Ping tries to determine which version of IP to use automatically.
a adds the IP source and destination addresses to each report.
f send messages as fast as possible (flood).
i sets the time between messages to be interval milliseconds, default 1000 ms.
l causes only lost messages to be reported.
n requests that a total of count messages be sent, default 32.
q suppresses any output (i.e. be quiet).
r randomizes the delay with a minimum extra delay of 0 ms and a maximum extra delay of the selected interval.
s sets the length of the message to be size bytes, ICMP header included. The size cannot be smaller than 32 or larger than 8192. The default is 64.
w sets the additional time in milliseconds to wait after all packets are sent.

Gping is a ping with a graphical display. It presents separate graphs for each destination specified.

The options are:
r display round trip time in seconds. This is the default.
l display percentage of lost messages. A message is considered lost if not replied to in 10 seconds. The percentage is an exponentially weighted average.
i sets the time between messages to be interval milliseconds, default 5000 ms.

Graphs can be dropped and added using the button 3 menu. Clicking button 1 on a datapoint displays the value of the datapoint and the time it was recorded.

Traceroute displays the IP addresses and average round trip times to all routers between the machine it is run on and dest. It does this by sending packets to dest with increasing times to live (TTL) in their headers. Each router that a packet expires at replies with an ICMP warning message. The options are:
d print debugging to standard error
n just print out IP numbers, don’t try to look up the names of the routers.
a make n attempts at each TTL value (default 3).
t set the starting TTL value to sttl (default 1).
print out a histogram of times from request to response at each TTL value. The histogram contains \textit{nbuck} buckets.

\textit{Hogports} announces on a range of ports to keep them from other processes. For example, to keep anyone from making a vncserver visible on the network mounted at /\texttt{net.alt}: 

\begin{verbatim}
ip/hogports /net.alt/tcp!*!5900-5950
\end{verbatim}

\textbf{SOURCE}
\begin{itemize}
\item /sys/src/cmd/ip/ping.c
\item /sys/src/cmd/ip/gping.c
\item /sys/src/cmd/ip/traceroute.c
\item /sys/src/cmd/ip/hogports.c
\end{itemize}

\textbf{SEE ALSO}
\begin{itemize}
\item ip(3)
\end{itemize}
NAME
plan9.ini – configuration file for PCs

SYNOPSIS
none

DESCRIPTION
When booting Plan 9 on a PC, the DOS program 9load(8) first reads a DOS file containing configuration information from the boot disk. This file, plan9.ini, looks like a shell script containing lines of the form

name=value

each of which defines a kernel or device parameter.

Blank lines and Carriage Returns (\r) are ignored. # comments are ignored, but are only recognised if # appears at the start of a line.

For devices, the generic format of value is

type=TYPE [port=N] [irq=N] [mem=N] [size=N] [dma=N] [ea=N]
specifying the controller type, the base I/O port of the interface, its interrupt level, the physical starting address of any mapped memory, the length in bytes of that memory, the DMA channel, and for Ethernets an override of the physical network address. Not all elements are relevant to all devices; the relevant values and their defaults are defined below in the description of each device.

The file is used by 9load and the kernel to configure the hardware available. The information it contains is also passed to the boot process, and subsequently other programs, as environment variables (see boot(8)). However, values whose names begin with an asterisk * are used by the kernel and are not converted into environment variables.

The following sections describe how variables are used.

ETHERNET
etherX=value

This defines an Ethernet interface. X, a unique monotonically increasing number beginning at 0, identifies an Ethernet card to be probed at system boot. Probing stops when a card is found or there is no line for etherX+1. After probing as directed by the etherX lines, any remaining Ethernet cards that can be automatically detected are added. Almost all cards can be automatically detected. For debugging purposes, automatic probing can be disabled by specifying the line *noetherprobe=. This automatic probing is only done by the kernel, not by 9load(8). Thus, if you want to load a kernel over the Ethernet, you need to specify an ether0 line so that 9load can find the Ethernet card, even if the kernel would have automatically detected it.

Some cards are software configurable and do not require all options. Unspecified options default to the factory defaults.

Known TYPES are

igbe The Intel 8254X Gigabit Ethernet controllers, as found on the Intel PRO/1000 adapters for copper (not fiber). Completely configurable.


rt18169 The Realtek 8169 Gigabit Ethernet controller. Completely configurable.

ga620 Netgear GA620 and GA620T Gigabit Ethernet cards, and other cards using the Alteon Acenic chip such as the Alteon Acenic fiber and copper cards, the DEC DEGPA-SA and the SGI Acenic. Completely configurable.

dp83820 National Semiconductor DP83820-based Gigabit Ethernet adapters, notably the D-Link DGE-500T. Completely configurable.

vgbe The VIA Velocity Gigabit Ethernet controller. Known to drive the VIA8237 (ABIT AV8), but at 100Mb/s full-duplex only.
m10g  The Myricom 10-Gigabit Ethernet 10G-PCIE-8A controller. Completely configurable. Can't boot through these due to enormous firmware loads.

i82598  The Intel 8259[89] 10-Gigabit Ethernet PCI-Express controllers. Completely configurable. Can't boot through these due to lack of a 9load driver.

i82557  Cards using the Intel 8255[789] Fast Ethernet PCI Bus LAN Controller such as the Intel EtherExpress PRO/1008. Completely configurable, no options need be given. If you need to force the media, specify one of the options (no value) 10BASE-T, 10BASE-2, 10BASE-5, 100BASE-TX, 10BASE-TFD, 100BASE-TXFD, 100BASE-T4, 100BASE-FX, or 100BASE-FXFD. Completely configurable.

2114x  Cards using the Digital Equipment (now Intel) 2114x PCI Fast Ethernet Adapter Controller, for example the Netgear FA310. Completely configurable, no options need be given. Media can be specified the same was as for the i82557. Some cards using the PNIC and PNIC2 near-clone chips may also work.

83815  National Semiconductor DP83815-based adapters, notably the Netgear FA311, Netgear FA312, and various SIS built-in controllers such as the SiS900. On the SIS controllers, the Ethernet address is not detected properly, specify it with an ea= attribute. Completely configurable.

rt18139  The Realtek 8139 Fast Ethernet controller. Completely configurable.

vt6102  The VIA VT6102 Fast Ethernet Controller (Rhine II).

smc91cxx  SMC 91cXX chip-based PCMCIA adapters, notably the SMC EtherEZ card.

elnk3  The 3COM Etherlink III series of cards including the 5x9, 59x, and 905 and 905B. Completely configurable, no options need be given. The media may be specified by setting media= to the value 10BaseT, 10Base2, 100BaseTX, 100BaseFX, aui, and miid. If you need to force full duplex, because for example the Ethernet switch does not negotiate correctly, just name the word (no value) fullDuplex or 100BASE-TXFD. Similarly, to force 100Mbit operation, specify force100. Port 0x110 is used for the little ISA configuration dance.

3c589  The 3COM 3C589 series PCMCIA cards, including the 3C562 and the 589E. There is no support for the modem on the 3C562. Completely configurable, no options need be given. Defaults are

    port=0x240  irq=10

    The media may be specified as media=10BaseT or media=10Base2.

ec2t  The Linksys Combo PCMCIA EthernetCard (EC2T), EtherFast 10/100 PCMCIA cards (PCMPC100) and integrated controllers (PCM100), the Netgear FA410TX 10/100 PCMCIA card and the Accton EtherPair—PCMCIA (EN2216). Completely configurable, no options need be given. Defaults are

    port=0x300  irq=9

    These cards are NE2000 clones. Other NE2000 compatible PCMCIA cards may be tried with the option

    id=string

    where string is a unique identifier string contained in the attribute memory of the card (see pcmcia(8)); unlike most options in plan9.in, this string is case-sensitive. The option dummyrr=[01] can be used to turn off (0) or on (1) a dummy remote read in the driver in such cases, depending on how NE2000 compatible they are.

ne2000  Not software configurable if ISA: PCI clones or supersets are software configurable; includes the Realtek 8029 clone used by Parallels. 16-bit card. Defaults are

    port=0x300  irq=2  mem=0x0a000  size=0x4000

    The option (no value) nodummyrr is needed on some (near) clones to turn off a dummy remote read in the driver.

amd79c970  The AMD PCnet PCI Ethernet Adapter (AM79C970). (This is the Ethernet adapter used by VMware.) Completely configurable, no options need be given.
wd8003 Includes WD8013 and SMC Elite and Elite Ultra cards. There are varying degrees of software configurability. Cards may be in either 8-bit or 16-bit slots. Defaults are port=0x280 irq=3 mem=0xD0000 size=0x2000
BUG: On many machines only the 16 bit card works.
sink A /dev/null for Ethernet packets — the interface discards sent packets and never receives any. This is used to provide a test bed for some experimental Ethernet bridging software.

wavelan Lucent Wavelan (Orinoco) IEEE 802.11b and compatible PCMCIA cards. Compatible cards include the Dell TrueMobile 1150 and the Linksys Instant Wireless Network PC Card. Port and IRQ defaults are 0x180 and 3 respectively.

These cards take a number of unique options to aid in identifying the card correctly on the 802.11b network. The network may be ad hoc or managed (i.e. use an access point):

mode=[adhoc, managed]
and defaults to managed. The 802.11b network to attach to (managed mode) or identify as (ad hoc mode), is specified by

esssid=string
and defaults to a null string. The card station name is given by

station=string
and defaults to Plan 9 STA. The channel to use is given by

channel=number
where number lies in the range 1 to 16 inclusive; the channel is normally negotiated automatically.

If the card is capable of encryption, the following options may be used:

crypt=[off, on]
and defaults to on.

keyN=string
sets the encryption key N (where N is in the range 1 to 4 inclusive) to string; this will also set the transmit key to N (see below). There are two formats for string which depend on the length of the string. If it is exactly 5 or 13 characters long it is assumed to be an alphanumeric key; if it is exactly 10 or 26 characters long the key is assumed to be in hex format (without a leading 0x). The lengths are checked, as is the format of a hex key.

txkey=number
sets the transmit key to use to be number in the range 1 to 4 inclusive. If it is desired to exclude or include unencrypted packets

clear=[off, on]
configures reception and defaults to inclusion.

The defaults are intended to match the common case of a managed network with encryption and a typical entry would only require, for example

essid=left-armpit key1=afish key2=calledraawaru
if the port and IRQ defaults are used. These options may be set after boot by writing to the device's ctf file using a space as the separator between option and value, e.g.

echo 'key2 1d8f65c9a52d83c8e4b43f94af' >/net/ether0/0/ctl
Card-specific power management may be enabled/disabled by

pm=[on, off]

wavelanpci PCI Ethernet adapters that use the same Wavelan programming interface. Currently the only tested cards are those based on the Intersil Prism 2.5 chipset.

DISKS, TAPES
(S)ATA controllers are autodetected.

usbX=type=uhci
usbX=type=ohci
This specifies the settings for a USB UHCI or OHCI controller. Like the Ethernet controllers, USB controllers are autodetected after scanning for the ones listed in plan9.ini. Thus, most systems will not need a usbX line. Also like the Ethernet controllers, USB autoprobing can be disabled by
specifying the line *nousbprobe=.

**scsiX=value**
This defines a SCSI interface which cannot be automatically detected by the kernel.
Known **TYPEs are**
aha1542
Adaptec 154x series of controllers (and clones). Almost completely configurable, only the
port=0x300
option need be given.
NCR/Symbios/LSI-Logic 53c8xx–based adapters and Mylex MultiMaster (Buslogic BT–) adapters
are automatically detected and need no entries.
By default, the NCR 53c8xx driver searches for up to 32 controllers. This can be changed by set-
ing the variable *maxsd53c8xx.
By default the Mylex driver resets SCSI cards by using both the hard reset and SCSI bus reset flags
in the driver interface. If a variable *noscsireset is defined, the SCSI bus reset flag is omitted.

**aoeif=list**
This specifies a space–separated list of Ethernet interfaces to be bound to the ATA–over-
Ethernet driver, aoe(3). For example, aoeif=ether0 ether1. Only interfaces on this list will
initially be accessible via AoE.

**aoedev=!#/aе/aoe/shelf.slot**
This specifies an ATA–over–Ethernet device accessible via the interfaces named in aoeif on AoE
shelf and slot to use as a root device for bootstrapping.

**AUDIO**

**audioX=value**
This defines a sound interface.
Known types are

**sb16** Sound Blaster 16.
**ess1688** A Sound Blaster clone.
The DMA channel may be any of 5, 6, or 7. The defaults are

```
port=0x220 irq=7 dma=5
```

**Uarts**
Plan 9 automatically configures COM1 and COM2, if found, as eia0 (port 0x3F8, IRQ4) and eia1
(port 0x2F8, IRQ3) respectively. These devices can be disabled by adding a line:

```
eiaX=disabled
```
This is typically done in order to reuse the IRQ for another device.
Plan 9 used to support various serial concentrators, including the TTC 8 serial line card and vari-
ous models in the Star Gate Avanstar series of intelligent serial boards. These are no longer sup-
ported; the much simpler Perle PCI–Fast4, PCI–Fast8, and PCI–Fast16 controllers have taken their
places. These latter cards are automatically detected and need no configuration lines.
The line serial=type=com can be used to specify settings for a PCMCIA modem.

**mouseport=value**
This specifies where the mouse is attached. **Value** can be

```
ps2  the PS2 mouse/keyboard port. The BIOS setup procedure should be used to configure the
machine appropriately.
```

```
ps2intellimouse
an Intellimouse on the PS2 port.
```

```
0  for COM1
1  for COM2
```

**modemport=value**
Picks the UART line to call out on. This is used when connecting to a file server over an async line.
Value is the number of the port.

console=value  params
This is used to specify the console device. The default value is cga; a number 0 or 1 specifies COM1 or COM2 respectively. A serial console is initially configured with the uart(3) configuration string b9600 18 pn s1, specifying 9600 baud, 8 bit bytes, no parity, and one stop bit. If params is given, it will be used to further configure the uart. Notice that there is no = sign in the params syntax. For example,

    console=0 b19200 po
would use COM1 at 19,200 baud with odd parity.

PC CARD
pccard0=disabled
   Disable probing for and automatic configuration of PC card controllers.

pcmciaX=type=XXX  irq=value
   If the default IRQ for the PCMCIA is correct, this entry can be omitted. The value of type is ignored.

pcmcia0=disabled
   Disable probing for and automatic configuration of PCMCIA controllers.

BOOTING
bootfile=value
   This is used to direct the actions of 9load(8) by naming the device and file from which to load the kernel.

rootdir=dir
rootspec=spec
   These are used by 9load(8) to identify the directory dir to make the root directory for the kernel, and the file system specifier spec (see mount in bind(2)) on which it can be found. These are usually used to test variant file systems for distributions, etc.

bootargs=value
   The value of this variable is passed to boot(8) by the kernel as the name of the root file system. It is typically used to specify additional arguments to pass to kfs(4) or ipconfig(8). For example, if the system is to run from a local kfs(4) partition, the definition might read

    bootargs=local!#S/sdC0/cache  kfs,bind

   See boot(8) for more.

nobootprompt=value
   Suppress the root from prompt and use value as the answer instead.

user=value
   Suppress the user prompt and use value as the answer instead.

debugfactotum=
   Causes boot(8) to start factotum with the -p option, so that it can be debugged.

factotumopts=options
   Causes boot(8) to start factotum with the given options, which must be a single word (i.e., contain no whitespace).

venti=value
   When booting from a local fossil server backed by a local or remote venti server, this variable specifies how to establish the connection to the venti server. See boot(8) for more.

cfs=value
   This gives the name of the file holding the disk partition for the cache file system, cfs(4). Extending the bootargs example, one would write cfs=#S/sdC0/cache.

bootdisk=value
   This deprecated variable was used to specify the disk used by the cache file system and other disk-resident services. It is superseded by bootargs and cfs.

partition=value
   This defines the partition table 9load(8) will examine to find disk partitioning information. By default, a partition table in a Plan 9 partition is consulted; if no such table is found, an old–Plan 9 partition table on the next-to-last or last sector of the disk is consulted. A value of new consults
only the first table, old only the second.
fs=a.b.c.d
auth=a.b.c.d
These specify the IP address of the file and authentication server to use when mounting a
network-provided root file system. They are used only if the addresses cannot be determined via
DHCP.

PROCESSOR
*norealmode=
The PC kernel switches the processor to 16-bit real mode to run BIOS interrupts, for example to
find the memory map or to enable VESA. This variable disables such switches.

*noe820scan=
When available, the PC kernel uses the BIOS E820 memory map to size memory. This variable disa-
bles the scan.

*maxmem=value
This defines the maximum physical address that the system will scan when sizing memory. By
default the PC operating system will scan up to 3.75 gigabytes (0xF0000000, the base of kernel
virtual address space), but setting *maxmem will limit the scan. *maxmem must be less than 3.75
gigabytes. This variable is not consulted if using the E820 memory map.

*kernelpercent=value
This defines what percentage of available memory is reserved for the kernel allocation pool. The
remainder is left for user processes. The default value is 30 on CPU servers, 60 on terminals with
less than 16MB of memory, and 40 on terminals with memories of 16MB or more. Terminals use
more kernel memory because *draw(3) maintains its graphic images in kernel memory. This depre-
cated option is rarely necessary in newer kernels.

*nomce=value
If machine check exceptions are supported by the processor, then they are enabled by default.
Setting this variable to 1 causes them to be disabled even when available.

*nomp=
A multiprocessor machine will enable all processors by default. Setting *nomp restricts the kernel
to starting only one processor and using the traditional interrupt controller.

*ncpu=value
Setting *ncpu restricts the kernel to starting at most value processors.

*pcimaxdno=value
This puts a limit on the maximum device number probed on a PCI bus (default 7). For example, a
value of 1 should suffice on a 'standard' motherboard with an AGP slot. This, and *pcimaxdno
below are rarely used and only on troublesome or suspect hardware.

*pcimaxdno=value
This puts a limit on the maximum device number probed on a PCI bus (default 31).

*nopcirouting=
Disable pci routing during boot. May solve interrupt routing problems on certain machines.

*nodumpstack=
Disable printing a stack dump on panic. Useful if there is only a limited cga screen available, oth-
erwise the textual information about the panic may scroll off.

ioexclude=value
Specifies a list of ranges of I/O ports to exclude from use by drivers. Ranges are inclusive on both
ends and separated by commas. For example:
ioexclude=0x330–0x337,0x430–0x43F

umbexclude=value
Specifies a list of ranges of UMB to exclude from use by drivers. Ranges are inclusive on both ends
and separated by commas. For example:
umbexclude=0xD1800–0xD3FFF

apm0=
This enables the “advanced power management” interface as described in *draw(3) and *draw(8).
The main feature of the interface is the ability to watch battery life (see stats(8)). It is not on by default because it causes problems on some laptops.

**VIDEO**

monitor=value

vgsizes=value

These are used not by the kernel but by termrc (see cpurc(8)) when starting vga(8).

*dpms=value

This is used to specify the screen blanking behavior of the MGA4xx video driver. Values are standby, suspend, and off. The first two specify differing levels of power saving; the third turns the monitor off completely.

**NVRAM**

nvram=file

nvrlen=length

nvroff=offset

This is used to specify an nvram device and optionally the length of the ram and read/write offset to use. These values are consulted by readnram (see authsrv(2)). The most common use of the nvram is to hold a seastore(1) password for use by factotum(4).

nvr=value

This is used by the WORM file server kernel to locate a file holding information to configure the file system. The file cannot live on a SCSI disk. The default is fd!0!plan9.nvr (sic), unless bootfile is set, in which case it is plan9.nvr on the same disk as bootfile. The syntax is either fd! unit! name or hd! unit! name where unit is the numeric unit id. This variant syntax is a vestige of the file server kernel’s origins.

**Multiple Configurations**

A plan9.ini file may contain multiple configurations, each within a block beginning with a line [tag]

A special block with the tag menu gives a list of blocks from which the user may interactively select the contents of plan9.ini. There may also be multiple blocks with the tag common which will be included in all selections; if any lines appear in plan9.ini before the first block, they are treated as a common block.

Within the menu block the following configuration lines are allowed:

menuitem=tag[, description]

The block identified by tag will appear in the presented menu. The menu entry will consist of the tag unless the optional description is given.

menudefault=tag[, timeout]

Identifies a default block to be given in the menu selection prompt. If the optional timeout is given (in seconds), the default block will be selected if there is no user input within the timeout period.

menuconsole=value[, baud]

Selects a serial console upon which to present the menu as no console or baud configuration information will have been processed yet (the plan9.ini contents are still to be decided...).

In response to the menu being printed, the user is prompted to select a menu item from the list. If the numeric response is followed by a p, the selected configuration is printed and the menu presented again.

The line

menuitem=tag

is prefixed to the selected configuration as an aid to user-level initialization scripts.

**EXAMPLES**

A representative plan9.ini:

```ini
% cat /n/c:/plan9.ini
ether0=type=3C509
mouseport=ps2
modemport=1
serial0=type=generic port=0x3E8 irq=5
```
monitor=445x
gvasize=1600x1200x8
%

Minimum CONFIG.SYS and AUTOEXEC.BAT files to use COM2 as a console:

% cat /n/c:/config.sys
SHELL=COMMAND.COM COM2 /P
% cat /n/c:/autoexec.bat
@ECHO OFF
PROMPT $p$g
PATH C:\DOS;C:\BIN
mode com2:96,n,8,1,p
SET TEMP=C:\IMP
%

Simple plan9.ini with multiple configurations:

    [menu]
    menuitem=vga, Plan 9 with VGA
    menuitem=novga, Plan 9 no automatic VGA
    menudefault=vga

    [vga]
    monitor=multisync135
    vgasize=1024x768x8

    [novga]

    [common]
    ether0=type=i82557
    audio0=type=sb16 port=0x220 irq=5 dma=1

With this, the following menu will be presented on boot:

    Plan 9 Startup Menu:
    ===================
    1. Plan 9 with VGA
    2. Plan 9 no automatic VGA
    Selection[default==1]:

Selecting item 1 generates the following plan9.ini to be used by the remainder of the boot-
strap process:

    menuitem=vga
    monitor=multisync135
    vgasize=1024x768x8
    ether0=type=i82557
    audio0=type=sb16 port=0x220 irq=5 dma=1

and selecting item 2:

    menuitem=novga
    ether0=type=i82557
    audio0=type=sb16 port=0x220 irq=5 dma=1

SEE ALSO
    9load(8), booting(8), boot(8)

BUGS

Being able to set the console device to other than a display is marginally useful on file servers;
MS-DOS and the programs which run under it are so tightly bound to the display that it is neces-
sary to have a display if any setup or reconfiguration programs need to be run. Also, the delay
before any messages appear at boot time is disconcerting, as any error messages from the BIOS
are lost.
This idea is at best an interesting experiment that needs another iteration.
NAME
pop3, imap4d - Internet mail servers

SYNOPSIS
upas/pop3 [ -d debugfile ] [ -a mailbox ] [ -r peeraddr ] [ -t tlscertfile ] [ -p ]
ip/imap4d [ -acpv ] [ -d smtpdomain ] [ -s servername ]

DESCRIPTION
These programs support remote access to mail across the Internet. All expect the network connection to be standard input, output, and error. They are normally started from scripts in /rc/bin/service (see listen(8)).

Pop3 provides access to a user’s mailboxes via the POP3 protocol. The options are:

- d create debugfile and write debugging output to it
- a causes pop3 to assume that it already authenticated and to read mailbox immediately
- r causes pop3 to create the file /mail/ratify/trusted/peeraddr#32 to allow subsequent SMTP sessions from that address. See ratfs(4) for details.
- t get the local TLS certificate from the file tlscertfile.
- p allow passwords in the clear for authenticating the connection

Imap4d provides access to a user’s mailboxes via the IMAP4rev1 protocol. Only files rooted in /mail/box/username/ are accessible. The list of subscribed mailboxes is contained in /mail/box/username/imap.subscribed, and initially contains only INBOX, IMAP’s name for the user’s mailbox. A shadow file, mailbox .imp, is created for each mailbox examined.

Imap4d’s options are:

a Assume the user is already authenticated. By default, the user must authenticate using CRAM-MD5 or securenet(8) challenge/response authentication.
c Allow plan 9 challenge response authentication.
p Allow login authentication. This option should only be enabled for servers using an encrypted connection, such as SSL, and when enabled, all non-encrypted connections should be disallowed. imap4d does not enforce this policy.
v Turn on verbose output to the debug file.
s The server’s name. If none is provided, cs (see ndb(8)) is queried or /env/sysname is used.
d The local mail domain. Defaults to the server /env/site in the mail server’s domain.

For both imap4d and pop3, the password used to authenticate the connection is the APOP secret held by keyfs(4) running on the authentication server.

FILES
/sys/log/imap4d debugging output
/mail/box/username/ mailbox
/mail/box/username/ mailbox .imp
/mail/box/username/imap.subscribed

SOURCE
/sys/src/cmd/upas/pop3
/sys/src/cmd/ip/imap4d

SEE ALSO
aliasmail(8), faces(1), filter(1), mail(1), marshal(1), mlmgr(1), nedmail(1), qer(8), rewrite(6), send(8), upasfs(4)

BUGS
Usually messages flagged for deletion with DELE are not actually deleted until the client sends a QUIT command to end the conversation. Pop3 implements a non-standard command SYNC that deletes messages flagged for deletion without ending the conversation.
NAME
ppp, pppoe, ptp, pptpd - point-to-point protocol

SYNOPSIS
ip/ppp [-CPScdfu] [-b baud] [-k keyspec] [-m mtu] [-M chatfile] [-p dev] [-x netmntpt] [-t modemcmd] [ local [ remote ]]
ip/pppoe [-D] [-A acname] [-S srvname] [-k keyspec] [-m mtu] [-x pppnetmntpt] [ ether ]
ip/pptp [-D] [-k keyspec] [-w window] [-x pppnetmntpt] [ server]

DESCRIPTION
The Point-to-Point Protocol is used to encapsulate Internet Protocol packets in IPv4 packets for
transfer over serial lines or other protocol connections. Ppp can run either as a client or, with the
-S option, as a server. The only differences between a client and a server is that the server will not
believe any local address the client tries to supply it and that the server always initiates the authen-
tication of the client.

With no option, ppp communicates with the remote system via standard input and output. This is
useful if a program wants to use ppp in a communications stream. However, the normal mode is
to specify a communications device, usually a serial line with a modem.

PPP supports the following options:
- b set the baud rate on the communications device
- f make PPP add HDLC framing. This is necessary when using PPP over a serial line or a TCP
  connection
- k add keyspec to the factotum(4) key pattern when looking for a user name and password for
  authentication; the default key pattern is proto=pass service=ppp
- m set the maximum transfer unit (default 1450)
- P use this as the primary IP interface; set the default route through this interface and write its
  configuration to /net/ndb
- p communicate over dev instead of standard I/O
- u before starting the PPP protocol with the remote end, shuttle bytes between the device and
  standard I/O until an EOF on standard input. This allows a user to start ppp and then type
  commands at a modem before ppp takes over
- S run as a server
- t before starting the PPP protocol, write modemcmd to the device
- x use the IP stack mounted at netmntpt
- M chat with the modem as specified in the chat file. Each line in the chat file contains a string
  that is transmitted to the modem and the response expected (e.g. 'AT' 'OK')
- c disallow packet compression
- C disallow ip header compression

If both the local and remote addresses are specified, don't ask the other end for either or believe it
if it supplies one. If either is missing, get it from the remote end.

Pppoe is a PPP over ethernet (PPPoE) client. It invokes ppp to start a PPP conversation which is tun-
neled in PPPoE packets on the ethernet device mounted at etherdir (default /net/ether0). The
pppoe -specific options are:
- A insist on an access concentrator named acname during PPPoE discovery
- S insist on a service named srvname during PPPoE discovery
- d write debugging output to standard error, and pass -d to ppp

The other options are relayed to ppp.
Ptp is a client for a PPTP encrypted tunnel. Server is the name of the server to dial. Ptp takes the same options as pppoe, except for the lack of a -m option and the addition of a -w option. The -w option specifies the local send window size (default 16) in packets.

Ptpd is the server side of a PPTP encrypted tunnel. Tcpdir is the directory of a TCP connection to the client. The TCP connection is used to control the tunnel while packets are sent back and forth using PPP inside of GRE packets. The options are:

d   write debugging output to standard error.
p   use the IP stack mounted at pppnetmtpt to terminate the PPP connection.
w   set the receive window to window.
D   drop fraction of the received packets. This is used for testing.

SOURCE
/sys/src/cmd/ip/PPP
/sys/src/cmd/ip/pptpd.c
/sys/src/cmd/ip/pppoe.c

SEE ALSO
   gre in ip(3)

BUGS
   Ppp should use factotum to execute the client side of the challenge–response protocol, but instead it reads a password from factotum and runs the protocol itself.
NAME
  prep, fdisk, format, mbr - prepare disks, floppies and flashes

SYNOPSIS
  disk/prep [-bcfnprw] [-a name]... [-s sectorsize] plan9partition
disk/fdisk [-abfprw] [-s sectorsize] disk
   file... ]
disk/mbr [-9] [-m mbrfile] disk

DESCRIPTION
A partition table is stored on a hard disk to specify the division of the physical disk into a set of 
logical units. On PCs, the partition table is stored at the end of the master boot record of the disk. 
Partitions of type 0x39 are Plan 9 partitions. The names of PC partitions are chosen by conven- 
tion from the type: dos, plan9, etc. Second and subsequent partitions of the same type on a 
given disk are given unique names by appending a number (or a period and a number if the name 
already ends in a number).

Plan 9 partitions (and Plan 9 disks on non-PCs) are themselves divided, using a textual partition 
table, called the Plan 9 partition table, in the second sector of the partition (the first is left for 
arbitrary architecture-specific boot data, such as PC boot blocks). The table is a sequence of lines of the 
format part name start end, where start and end name the starting and ending sector. Sector 0 
is the first sector of the Plan 9 partition or disk, regardless of its position in a larger disk. Partition 
extents do not contain the ending sector, so a partition from 0 to 5 and a partition from 5 to 10 do 
ot overlap.

The Plan 9 partition often contains a number of conventionally named subpartitions. They include:

9fat    A small FAT file system used to hold configuration information (such as plan9.ini 
        and plan9.nvr) and kernels. This typically begins in the first sector of the partition, 
        and contains the partition table as a "reserved" sector. See the discussion of the -r 
        option to format.
arenas  A venti(8) arenas partition.
bloom  A venti(8) bloom-filter partition.
cache  A cfs(4) file system cache.
fossil  A fossil(4) file system.
fs      A kfs(4) file system.
fscfg   A one-sector partition used to store an fs(3) configuration.
isect   A venti(8) index section.
nvram   A one-sector partition used to simulate non-volatile RAM on PCs.
other   A non-archived fossil(4) file system.
swap    A swap(8) swap partition.

Fdisk edits the PC partition table and is usually invoked with a disk like /dev/sdC0/data as its 
argument, while prep edits the Plan 9 partition table and is usually invoked with a disk partition 
like /dev/sdC0/plan9 as its argument. Fdisk works in units of disk "cylinders"; the cylinder 
size in bytes is printed when fdisk starts. Prep works in units of disk sectors, which are almost 
always 512 bytes. Fdisk and prep share most of their options:

-a    Automatically partition the disk. Fdisk will create a Plan 9 partition in the largest unused 
      area on the disk, doing nothing if a Plan 9 partition already exists. If no other partition on 
      the disk is marked active (i.e. marked as the boot partition), fdisk will mark the new partition 
      active. Prep’s -a flag takes the name of a partition to create. (See the list above for parti- 
      tion names.) It can be repeated to specify a list of partitions to create. If the disk is cur- 
      rently unpartitioned, prep will create the named partitions on the disk, attempting to use the 
      entire disk in a sensible manner. The partition names must be from the list given above.
-b    Start with a blank disk, ignoring any extant partition table.
-p    Print a sequence of commands that when sent to the disk device’s c.t1 file will bring the par- 
      tition table information kept by the sd(3) driver up to date. Then exit. Prep will check to see 
      if it is being called with a disk partition (rather than an entire disk) as its argument; if so, it
will translate the printed sectors by the partition’s offset within the disk. Since *fdisks* operates on a table of unnamed partitions, it assigns names based on the partition type (e.g., plan9, dos, ntfs, linux, linuxswap) and resolves collisions by appending a numbered suffix. (e.g., dos, dos.1, dos.2).

- `r` In the absence of the `-p` and `-w` flags, *prep* and *fdisks* enter an interactive partition editor; the `-r` flag runs the editor in read-only mode.

- `s` *sectorsize*

  Specify the disk’s sector size. In the absence of this flag, *prep* and *fdisks* look for a disk c.t.l file and read it to find the disk’s sector size. If the c.t.l file cannot be found, a message is printed and a sector size of 512 bytes is assumed.

- `w` Write the partition table to the disk and exit. This is useful when used in conjunction with `-a` or `-b`.

If neither the `-p` flag nor the `-w` flag is given, *prep* and *fdisks* enter an interactive partition editor that operates on named partitions. The PC partition table distinguishes between primary partitions, which can be listed in the boot sector at the beginning of the disk, and secondary (or extended) partitions, arbitrarily many of which may be chained together in place of a primary partition. Primary partitions are named *pn*, secondary partitions *sn*. The number of primary partitions plus number of contiguous chains of secondary partitions cannot exceed four.

The commands are as follows. In the descriptions, read “`sector`” as “cylinder” when using *fdisks*.

```
a name [ start [ end ] ]
```

Create a partition named *name* starting at sector offset *start* and ending at offset *end*. The new partition will not be created if it overlaps an extant partition. If *start* or *end* are omitted, *prep* and *fdisks* will prompt for them. In *fdisks*, the newly created partition has type “PLAN9” to set a different type, use the `t` command (q.v.). *Start* and *end* may be expressions using the operators +, -, *, /, numeric constants, and the pseudo-variables . and $. At the start of the program, . is set to zero; each time a partition is created, it is set to the end sector of the new partition. It can also be explicitly set using the . command. When evaluating *start*, $ is set to one past the last disk sector. When evaluating *end*, $ is set to the maximum value that *end* can take on without running off the disk or into another partition. Numeric constants followed by k, m, g, or t (or upper-case equivalents) are scaled to the respective size in kilo-, mega-, giga-, or tera-bytes. Finally, the expression `n%` evaluates to `(n×disksize)/100`. As examples, a .+20% creates a new partition starting at . that takes up a fifth of the disk, a . +21G creates a new partition starting at . that takes up 21 gigabytes (21×230 bytes), and a 1000 $ creates a new partition starting at sector 1000 and extending as far as possible.

```
. newdot
```

Set the value of the variable . to newdot, which is an arithmetic expression as described in the discussion of the `a` command.

```
d name
```

Delete the named partition.

```
h
```

Print a help message listing command synopses.

```
p
```

Print the disk partition table. Unpartitioned regions are also listed. The table consists of a number of lines containing partition name, beginning and ending sectors, and total size. A ‘ ‘ is prefixed to the names of partitions whose entries have been modified but not written to disk. *Fdisk* adds to the end of each line a textual partition type, and places a ‘*’ next to the name of the active partition (see the A command below).

```
P
```

Print the partition table in the format accepted by the disk’s c.t.l file, which is also the format of the output of the `-p` option.

```
w
```

Write the partition table to disk. *Prep* will also inform the kernel of the changed partition table. The write will fail if any programs have any of the disk’s partitions open. If the write fails (for this or any other reason), *prep* and *fdisks* will attempt to restore the partition table to its former state.

```
q
```

Quit the program. If the partition table has been modified but not written, a warning is printed. Typing q again will quit the program.
Fdisk also has the following commands.

A name Set the named partition active. The active partition is the one whose boot block is used when booting a PC from disk.

e Print the names of empty slots in the partition table, i.e., the valid names to use when creating a new partition.

t [ type ] Set the partition type. If it is not given, Fdisk will display a list of choices and then prompt for it.

Format prepares for use the floppy diskette or hard disk partition in the file named disk, for example /dev/fd0disk or /dev/sdC0/9fat. The options are:

--f Do not physically format the disc. Used to install a FAT file system on a previously formatted disc. If disk is not a floppy device, this flag is a no-op.

--t specify a density and type of disk to be prepared. The possible types are:

3½DD 3½” double density, 737280 bytes
3½HD 3½” high density, 1474560 bytes
5¼DD 5¼” double density, 368640 bytes
5¼HD 5¼” high density, 1146880 bytes

hard fixed disk

The default when disk is a floppy drive is the highest possible on the device. When disk is a regular file, the default is 3½HD. When disk is an sd(3) device, the default is hard.

--d initialize a FAT file system on the disk.

--b use the contents of bootblock as a bootstrap block to be installed in sector 0.

The remaining options have effect only when --d is specified:

--c use a FAT cluster size of csize sectors when creating the FAT.

--l add a label when creating the FAT file system.

--r mark the first nresrv sectors of the partition as “reserved”. Since the first sector always contains the FAT parameter block, this really marks the nresrv–1 sectors starting at sector 1 as “reserved”. When formatting the 9fat partition, --r 2 should be used to jump over the partition table sector.

Again under --d, any files listed are added, in order, to the root directory of the FAT file system. The files are contiguous allocated. If a file is named 9load, it will be created with the SYSTEM attribute set so that dossrv(4) keeps it contiguous when modifying it.

Format checks for a number of common mistakes; in particular, it will refuse to format a 9fat partition unless --r is specified with nresrv larger than two. It also refuses to format a raw sd(3) partition that begins at offset zero in the disk. (The beginning of the disk should contain an Fdisk partition table with master boot record, not a FAT file system or boot block.) Both checks are disabled by the --x option. The --v option prints debugging information.

The file /386/pbs is an example of a suitable bfile to make the disk a boot disk. It gets loaded by the BIOS at 0x7c00, reads the first sector of the root directory into address 0x7e00, and looks for a directory entry named 9LOAD. If it finds such an entry, it uses single sector reads to load the file into address 0x10000 and then jumps to the loaded file image. The file /386/pbs1ba is similar, but because it uses LBA addressing (not supported by older BIOSes), it can access more than the first 8.5GB of the disk.

Mbr installs a new boot block in sector 0 (the master boot record) of a disk such as /dev/sdC0/data. If mbfile contains more than one sector of ‘boot block’, the rest will be copied into the first track of the disk, if it fits. This boot block should not be confused with the boot block used by format, which goes in sector 0 of a partition. Typically, the boot block in the master boot record scans the PC partition table to find an active partition and then executes the boot block for that partition. The partition boot block then loads a bootstrap program such as 9load(8), which then loads the operating system. If MS-DOS or Windows 9[58] is already installed on your hard disk, the master boot record already has a suitable boot block. Otherwise,
/386/mbr is an appropriate mbrfile. It detects and uses LBA addressing when available from the BIOS (the same could not be done in the case of pbs due to space considerations). If the mbrfile is not specified, a boot block is installed that prints a message explaining that the disk is not bootable. The -9 option initialises the partition table to consist of one plan9 partition which spans the entire disc starting at the end of the first track.

EXAMPLES
Initialize the kernel disk driver with the partition information from the FAT boot sectors. If Plan 9 partitions exist, pass that partition information as well.

```bash
for(disk in /dev/sd??) {
    if(test -f $disk/data && test -f $disk/ctl)
        disk/fdisk −b $disk/data >$disk/ctl
    for(part in $disk/plan9*)
        if(test −f $part)
            disk/prep −p $part >$disk/ctl
}
```

Create a Plan 9 boot floppy on a previously formatted diskette.

```bash
disk/format −b /386/pbs −df /dev/fd0disk \
  /386/9load /tmp/plan9.ini /386/9pcf.gz
```

Initialize the blank hard disk /dev/sdC0/data.

```bash
disk/mbr −m /386/mbr /dev/sdC0/data
disk/fdisk −baw /dev/sdC0/data
disk/prep −bw −a(9fat nvram fossil cache swap) /dev/sdC0/plan9
disk/format −b /386/pbslba −d −r 2 /dev/sdC0/9fat \
  /386/9load /386/9pcf /tmp/plan9.ini
```

Create a bootable USB disk or flash−memory device to be booted via the BIOS and containing only a FAT Plan 9 partition.

```bash
mount /srv/usb /n/usb
ls −d /n/usb/sdU* # note the name, normally sdU0.0
disk/partfs /n/usb/sdU0.0/data
cd /dev/sdXX
disk/mbr −m /386/mbr data
disk/fdisk −baw data
disk/prep −bw −a(9fat nvram fossil) plan9
cp /386/9loadusb /tmp/9load # force format to use the name ‘9load’
disk/format −b /386/pbslba −d −r 2 9fat /tmp/9load \
  /386/9pcppuf /tmp/plan9.ini
```

FILES
/386/mbr
/386/mbr.bootmgr  self-configuring 'smart boot manager’

SOURCE
/sys/src/cmd/disk/prep
/sys/src/boot/pc
/n/sources/extra/bootmgr.tgz

```
source for /386/mbr.bootmgr; compilable on (l)unix
```

SEE ALSO
floppy(3), sd(3), usb(4), 9load(8), partfs(8)

BUGS
Format can create FAT12 and FAT16 file systems, but not FAT32 file systems. The boot block can only read from FAT12 and FAT16 file systems.

If prep −p doesn't find a Plan 9 partition table, it will emit commands to delete all extant partitions. Similarly, fdisk −p will delete all partitions, including data, if there are no partitions defined in the MBR.
NAME
qer, runq - queue management for spooled files

SYNOPSIS
qer [-q subdir] [-E file] root tag reply args

DESCRIPTION
Qer creates a control and a data file in a queue directory. The control file contents consist of the tag, reply, and args separated by spaces. The data file contains the standard input to qer. The files are created in the directory root/subdir, where subdir is the argument to -q if present, else the contents of /dev/user. The names of the control and data files differ only in the first character which is 'C' and 'D' respectively. Mkttemp(2) is used to create the actual names of the control and data file.

Some commands, such as fax (see telco(4)), must queue more files than just the data file. Each file following a -E flag is copied into the queue directory. The names of the copies differ from the name of the data file only in the first character. The first one is starts with 'F', the second 'C', etc.

Runq processes the files queued by qer. Without the -a option, runq processes all requests in the directory root/subdir, where subdir is the argument to -q if present, else the contents of /dev/user. With the -a it processes all requests. Each request is processed by executing the command cmd with the contents of the control file as its arguments, the contents of the data file as its standard input, and standard error appended to the error file /dev/null.

The action taken by runq depends on the return status of cmd. If cmd returns a null status, the processing is assumed successful and the control, data, and error files are removed. If cmd returns an error status containing the word Retry, the files are left to be reprocessed at a later time. For any other status, an error message is mailed to the requester and the files are removed. Runq uses the reply field in the control file as a mail address to which to send an error notification. The notification contains the contents of the control file to identify the failed request.

To avoid reprocessing files too often, the following algorithm is used: a data file younger than one hour will not be processed if its error file exists and was last modified within the preceding 10 minutes. A data file older than one hour will not be processed if its error file exists and was last modified within the preceding hour. The -E flag causes all files to be reprocessed regardless of the file times.

The -R flag instructs runq never to give up on a failed queue job, instead leaving it in the queue to be retried.

The -d option causes debugging output on standard error describing the progress through the queues.

The -t flags specifies the number of hours that retries will continue after a send failure. The default is 48 hours.

The -r flag limits the number of files that are processed in a single pass of a queue. Runq accumulates the entire directory containing a queue before processing any files. When a queue contains many files and the system does not have enough memory, runq exits without making progress. This flag forces runq to process the directory in chunks, allowing the queue to be drained incrementally. It is most useful in combination with the -q flag.

The -s, -n, and -l flags are only meaningful with the -a flag. They control amount of parallelism that is used when sweeping all of the queues. The argument following the -n flag specifies the number of queues that are swept in parallel; the default is 50. The argument following the -l flag specifies the total number of queues that are being swept. By default, there is no limit. The number of active sweeps is cumulative over all active executions of runq. The -s flag forces each queue directory to be processed by exactly one instance of runq. This is useful on systems that connect to slow external systems and prevents all the queue sweeps from piling up trying to process a few slow systems.

Runq is often called from cron(8) by an entry such as
0,10,20,30,40,50 * * * * kremvax
runq -a /mail/queue /mail/lib/remotemail

The entry must be a single line; it is folded here only so it fits on the page.

FILES
  
  root/user
  root/user/D.XXXXXX
  root/user/C.XXXXXX
  root/user/E.XXXXXX
  root/user/[F−Z].XXXXXX

queue directory for user
data file
control file
error file
secondary data files

SOURCE
/sys/src/cmd/upas/q

SEE ALSO
-mail(1)
NAME
reboot – reboot the system upon loss of remote file server connection

SYNOPSIS
aux/reboot [file]

DESCRIPTION
Reboot stats file, default /$cputype/lib, once every five minutes. If the stat fails, rather than
timing out, reboot reboots the system. This is used to restart diskless cpu servers whenever their
file server connection is broken.

SOURCE
/sys/src/cmd/aux/reboot.c
NAME
applychanges, applylog, compactdb, updatedb - simple client–server replica management

SYNOPSIS
replica/compactdb db
replica/updatedb [−cl][−p proto][−r root][−t now n][−u uid][−x path]... db
replica/applylog [−nv] [−c name]... [−s name]... clientdb clientroot serverroot [path ...]
replica/applychanges [−nv] [−p proto] [−x path]... clientdb clientroot serverroot [path ...]

DESCRIPTION
These four tools collectively provide simple log-based client–server replica management. The shell scripts described in replica(1) provide a more polished interface.

Both client and server maintain textual databases of file system metadata. Each line is of the form

path mode uid gid mtime length

Later entries for a path supersede previous ones. A line with the string REMOVED in the mode field annuls all previous entries for that path. The entries in a file are typically kept sorted by path but need not be. These properties facilitate updating the database atomically by appending to it. Compactdb reads in a database and writes out an equivalent one, sorted by path and without outdated or annulled records.

A replica is further described on the server by a textual log listing creation and deletion of files and changes to file contents and metadata. Each line is of the form:

time gen verb path serverpath mode uid gid mtime length

The time and gen fields are both decimal numbers, providing an ordering for log entries so that incremental tools need not process the whole log each time they are run. The verb, a single character, describes the event: addition of a file (a), deletion of a file (d), a change to a file’s contents (c), or a change to a file’s metadata (m). Path is the file path on the client; serverpath the path on the server (these are different when the optional fifth field in a proto file line is given; see proto(2)). Mode, uid, gid, and mtime are the files metadata as in the Dir structure (see stat(5)). For deletion events, the metadata is that of the deleted file. For other events, the metadata is that after the event.

Updatedb scans the file system rooted at root for changes not present in db, noting them by appending new entries to the database and by writing log events to standard output. The −c option causes updatedb to consider only file and metadata changes, ignoring file additions and deletions. By default, the log events have time set to the current system time and use incrementing gen numbers starting at 0. The −t option can be used to specify a different time and starting number. If the −u option is given, all database entries and log events will use uid rather than the actual uids. The −x option (which may be specified multiple times) excludes the named path and all its children from the scan. If the −l option is given, the database is not changed and the time and gen fields are omitted from the log events; the resulting output is intended to be a human-readable summary of file system activity since the last scan.

Applylog is used to propagate changes from server to client. It applies the changes listed in a log (read from standard input) to the file system rooted at clientroot, copying files when necessary from the file system rooted at serverroot. By default, applylog does not attempt to set the uid on files; the −u flag enables this. Applylog will not overwrite local changes made to replicated files. When it detects such conflicts, by default it prints an error describing the conflict and takes no action. If the −c flag is given, applylog still takes no action for files beginning with the given names, but does so silently and will not report the conflicts in the future. (The conflict is resolved in favor of the client.) The −s is similar but causes applylog to overwrite the local changes. (The conflict is resolved in favor of the server.)

Applychanges is, in some sense, the opposite of applylog; it scans the client file system for changes, and applies those changes to the server file system. Applychanges will not overwrite remote changes made to replicated files. For example, if a file is copied from server to client and subsequently changed on both server and client, applychanges will not copy the client’s new
version to the server, because the server also has a new version. *Applychanges* and *applylog* detect the same conflicts; to resolve conflicts reported by *applychanges*, invoke *applylog* with the 

-`c` or -`s` flags.

**EXAMPLE**

One might keep a client kfs file system up-to-date against a server file system using these tools. First, connect to a CPU server with a high-speed network connection to the file server and scan the server file system, updating the server database and log:

```plaintext
repl=$home/lib/replica
proto=/sys/lib/sysconfig/proto/portproto
db=$repl/srv.portproto.db
log=$repl/srv.portproto.log
```

```bash
9fs $fs
replica/updatedb -p $proto -r /n/$fs -x $repl $db >>$log
replica/compactdb $db >/tmp/a && mv /tmp/a $db
```

Then, update the client file system:

```bash
repl=$home/lib/replica
db=$repl/cli.portproto.db
log=$repl/srv.portproto.log
```

```bash
9fs $fs
9fs kfs
replica/applylog $db /n/kfs /n/$fs <$log
replica/compactdb $db >/tmp/a && mv /tmp/a $db
```

The `$repl` directory is excluded from the sync so that multiple clients can each have their own local database. The shell scripts in `/rc/bin/replica` are essentially a further development of this example.

The Plan 9 distribution update program operates similarly, but omits the first scan; it is assumed that the Plan 9 developers run scans manually when the distribution file system changes. The manual page `replica(1)` describes this in full.

**SEE ALSO**

`replica(1)`

**BUGS**

These tools assume that *mtime* combined with *length* is a good indicator of changes to a file’s contents.
NAME
rsagen, rsafill, asnl2rsa, rsa2pub, rsa2ssh, rsa2x509 - generate and format rsa keys

SYNOPSIS
rsagen [-b nbits] [-t tag]
rsafill [file]
asnl2rsa [-t tag] [file]
rsa2pub [file]
rsa2ssh [file]
rsa2x509 [-e expirerime] certinfo [file]

DESCRIPTION
Plan 9 represents an RSA key as an attribute-value pair list prefixed with the string key: this is the
generic key format used by factotum(4). A full RSA private key has the following attributes:

proto must be rsa
size the number of significant bits in n
ek the encryption exponent
n the product of p and q
!dk the decryption exponent
!p a large prime
!q another large prime
!kp, !kq, !c2
parameters derived from the other attributes, cached to speed decryption

All the numbers are in hexadecimal except size , which is decimal. An RSA public key omits the
attributes beginning with ! . A key may have other attributes as well (for example, a service
attribute identifying how this key is typically used), but to these utilities such attributes are merely
comments.

For example, a very small (and thus insecure) private key and corresponding public key might be:

    key proto=rsa size=8 ek=7 n=8F !dk=67 !p=B !q=D !kp=3 !kq=7 !c2=6
    key proto=rsa size=8 ek=7 n=8F

Note that the order of the attributes does not matter.

Rsagen prints a randomly generated RSA private key whose n has exactly nbits (default 1024) sig-
nificant bits. If tag is specified, it is printed between key and proto=rsa; typically, tag is a
sequence of attribute-value comments describing the key.

Rsafill reads a private key, recomputes the !kp, !kq, and !c2 attributes if they are missing, and
prints a full key.

Asnl2rsa reads an RSA private key stored as ASN.1 encoded in the binary Distinguished Encoding
Rules (DER) and prints a Plan 9 RSA key, inserting tag exactly as rsagen does. ASN.1/DER is a
popular key format on Unix and Windows; it is often encoded in text form using the Privacy Enhanced
Mail (PEM) format in a section labeled as “RSA PRIVATE KEY.” The command:

    auth/pemdecode 'RSA PRIVATE KEY' | auth/asnl2rsa

extracts the key section from a textual ASN.1/DER/PEM key into binary ASN.1/DER format and then
converts it to a Plan 9 RSA key.

Rsa2pub reads a Plan 9 RSA public or private key, removes the private attributes, and prints the
resulting public key. Comment attributes are preserved.

Rsa2ssh reads a Plan 9 RSA public or private key and prints the public portion in the format used
by SSH: three space-separated decimal numbers size, ek, and n. For compatibility with external
SSH implementations, the public keys in /sys/lib/ssh/keyring and
$home/lib/keyring are stored in this format.
RSA(8)

Rsax509 reads a Plan 9 RSA private key and writes a self-signed X.509 certificate encoded in ASN.1/DER format to standard output. (Note that ASN.1/DER X.509 certificates are different from ASN.1/DER private keys). The certificate uses the current time as its start time and expires expiretime seconds (default 3 years) later. It contains the public half of the key and includes certinfo as the issuer/subject string (also known as a “Distinguished Name”). This info is typically in the form:

C=US ST=NJ L=07974 O=Lucent OU='Bell Labs' CN=G.R.Emlin

The X.509 ASN.1/DER format is often encoded in text using a PEM section labeled as a "CERTIFICATE." The command:

```
auth/rsax509 'C=US OU='Bell Labs'' file |
auth/pemencode CERTIFICATE
```

generates such a textual certificate. Applications that serve TLS-encrypted sessions (for example, httpd(8), pop3(8), and tlssrv(8)) expect certificates in ASN.1/DER/PEM format.

EXAMPLES

Generate a fresh key and use it to start a TLS-enabled web server:

```
auth/rsagen -t 'service=tls owner=*' >key
auth/rsax509 'C=US CN=*.cs.bell-labs.com' key |
auth/pemencode CERTIFICATE >cert
cat key >/mnt/factotum/ctl
ip/httpd/httpd -c cert
```

Generate a fresh key and configure a remote Unix system to allow use of that key for logins:

```
auth/rsagen -t 'service=ssh' >key
auth/rsassh key | ssh unix 'cat >>.ssh/authorized_keys'
cat key >/mnt/factotum/ctl
ssh unix
```

SOURCE

/sys/src/cmd/auth

SEE ALSO

factotum(4), pem(8), ssh(1)

BUGS

There are too many key formats.
NAME
    rtcp, stcp – measure TCP bandwidth

SYNOPSIS

DESCRIPTION
    Rtcp and stcp together measure TCP bandwidth between the two machines on which they are run.
    Rtcp should be run first on the target system, then stcp on the local system. When r tcp is interrupted or exits, stcp will print the average megabytes-per-second transferred during the session. By default, the TCP service used is ttcp (port 5117).
    Rtcp’s options are:
        -a Use an alternate announce-str.
        -d Enable verbose debugging output.
        -f Use floating point to compute throughput.
        -i Print throughput every interval seconds (default 1).
        -l Read readsize bytes at a time instead of the default 8192.
        -L Write log file.
        -t Shuffle bytes read one byte to the left.
        -T Receive for secs seconds only.
        -v Print more.
    Stcp’s options are:
        -f Read from filename instead of standard input.
        -F Set the TCP maximum fragment size to maxfragsize.
        -l Write writesize bytes at a time instead of the 8192.
        -s Wait for the receiving process to exit.
        -T Transmit only totsize bytes.
        -v Print more.

EXAMPLES
    Measure bandwidth to go. On go, run
      ip/rtcp
    On the local system, run
      ip/stcp -T10000000 go

SOURCE
    /sys/src/cmd/ip/rtcp

SEE ALSO
    ping(8)
NAME
scanmail, testscan - spam filters

SYNOPSIS
upas/scanmail [ options ][ qer-args ] root mail sender system rcpt-list
upas/testsca[n [−avd][−p patfile][filename]

DESCRIPTION
Scanmail accepts a mail message supplied on standard input, applies a file of patterns to a portion of it, and dispatches the message based on the results. It exactly replaces the generic queuing command qer(8) that is executed from the rc(1) script /mail/lib/qmail in the mail processing pipeline. Associated with each pattern is an action in order of decreasing priority:

dump the message is deleted and a log entry is written to /sys/log/smtpd
hold the message is placed in a queue for human inspection
log a line containing the matching portion of the message is written to a log

If no pattern matches or only patterns with an action of log match, the message is accepted and scanmail queues the message for delivery. Scanmail meshes with the blocking facilities of smtpd(6) to provide several layers of filtering on gateway systems. In all cases the sender is notified that the message has been successfully delivered, leaving the sender unaware that the message has been potentially delayed or deleted.

Scanmail accepts the arguments of qer(8) as well as the following:

−c Save a copy of each message in a randomly-named file in directory /mail/copy.
−d Write debugging information to standard error.
−h Queue held messages by sending domain name. The −q option must specify a root directory; messages are queued in subdirectories of this directory. If the −h option is not specified, messages are accumulated in a subdirectory of /mail/queue.hold named for the contents of /dev/user, usually none.
−n Messages are never held for inspection, but are delivered. Also known as vacation mode.
−p filename Read the patterns from filename rather than /mail/lib/patterns.
−q holdroot Queue deliverable messages in subdirectories of holdroot. This option is the same as the −q option of qer(8) and must be present if the −h option is given.
−s Save deleted messages. Messages are stored, one per randomly-named file, in subdirectories of /mail/queue.dump named with the date.
−t Test mode. The pattern matcher is applied but the message is discarded and the result is not logged.
−v Print the highest priority match. This is useful with the −t option for testing the pattern matcher without actually sending a message.

Testsca[n is the command line version of scanmail. If filename is missing, it applies the pattern set to the message on standard input. Unlike scanmail, which finds the highest priority match, testsca[n prints all matches in the portion of the message under test. It is useful for testing a pattern set or implementing a personal filter using the pipeto file in a user's mail directory. Testsca[n accepts the following options:

−a Print matches in the complete input message
−d Enable debug mode
−v Print the message after conversion to canonical form (q.v.).
−p filename Read the patterns from filename rather than /mail/lib/patterns.

Canonicalization
Before pattern matching, both programs convert a portion of the message header and the beginning of the message to a canonical form. The amount of the header and message body processed are set by compile-time parameters in the source files. The canonicalization process converts letters to lower-case and replaces consecutive spaces, tabs and newline characters with a single space. HTML commands are deleted except for the parameters following A HREF, IMG SRC, and
IMG BORDER directives. Additionally, the following MIME escape sequences are replaced by their ASCII equivalents:

<table>
<thead>
<tr>
<th>Escape Seq</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>=2e</td>
<td>.</td>
</tr>
<tr>
<td>=2f</td>
<td>/</td>
</tr>
<tr>
<td>=20</td>
<td>&lt;space&gt;</td>
</tr>
<tr>
<td>=3d</td>
<td>=</td>
</tr>
</tbody>
</table>

and the sequence =<newline> is elided. *Scanmail* assembles the sender, destination domain and recipient fields of the command line into a string that is subjected to the same canonical processing. Following canonicalization, the command line and the two long strings containing the header and the message body are passed to the matching engine for analysis.

**Pattern Syntax**

The matching engine compiles the pattern set and matches it to each canonicalized input string. Patterns are specified one per line as follows:

```{*:action: pattern-spec {~ ~ override... ~ ~ override}}
```

On all lines, a # introduces a comment; there is no way to escape this character.

Lines beginning with * contain a `pattern-spec` that is a string; otherwise, the the `pattern-spec` is a regular expression in the style of `regexp(6)`. Regular expression matching is many times less efficient than string matching, so it is wiser to enumerate several similar strings than to combine them into a regular expression. The *action* is a keyword terminated by a : and separated from the pattern by optional white-space. It must be one of the following:

- **dump** if the pattern matches, the message is deleted. If the –s command line option is set, the message is saved.
- **hold** if the pattern matches, the message is queued in a subdirectory of /mail/queue.hold for manual inspection. After inspection, the queue can be swept manually using `rundq` (see `qer(8)` to deliver messages that were inadvertently matched.
- **header** this is the same as the hold action, except the pattern is only applied to the message header. This optimization is useful for patterns that match header fields that are unlikely to be present in the body of the message.
- **line** the sender and a section of the message around the match are written to the file /sys/log/lines. The message is always delivered.
- **loff** patterns of this type are applied only to the canonicalized command line. When a match occurs, all patterns with line actions are disabled. This is useful for limiting the size of the log file by excluding repetitive messages, such as those from mailing lists.

Patterns are accumulated into pattern sets sharing the same action. The matching engine applies the dump pattern set first, then the header and hold pattern sets, and finally the line pattern set. Each pattern set is applied three times: to the canonicalized command line, to the message header, and finally to the message body. The ordering of patterns in the pattern file is insignificant.

The `pattern-spec` is a string of characters terminated by a newline, # or override indicator, ~ ~. Trailing white-space is deleted but patterns containing leading or trailing white-space can be enclosed in double-quote characters. A pattern containing a double-quote must be enclosed in double-quote characters and preceded by a backslash. For example, the pattern

```
"this is not "spam"
```

matches the string *this is not "spam"*. The `pattern-spec` is followed by zero or more override strings. When the specific pattern matches, each override is applied and if one matches, it cancels the effect of the pattern. Overrides must be strings; regular expressions are not supported. Each override is introduced by the string ~ ~ and continues until a subsequent ~ ~, # or newline, white-space included. A ~ ~ immediately followed by a newline indicates a line continuation and further overrides continue on the following line. Leading white-space on the continuation line is ignored. For example,
*hold:  sex.com~~essex.com~~sussex.com~~sysex.com~~
        lasex.com~~cse.psu.edu!owner−9fans
matches all input containing the string sex.com except for messages that also contain the
strings in the override list. Often it is desirable to override a pattern based on the name of the
sender or recipient. For this reason, each override pattern is applied to the header and the com-
mand line as well as the section of the canonicalized input containing the matching data. Thus a
pattern matching the command line or the header searches both the command line and the header
for overrides while a match in the body searches the body, header and command line for overrides.
The structure of the pattern file and the matching algorithm define the strategy for detecting and
filtering unwanted messages. Ideally, a hold pattern selects a message for inspection and if it is
determined to be undesirable, a specific dump pattern is added to delete further instances of the
message. Additionally, it is often useful to block the sender by updating the smtpd control file.
In this regime, patterns with a dump action, generally match phrases that are likely to be unique.
Patterns that hold a message for inspection match phrases commonly found in undesirable mate-
rival and occasionally in legitimate messages. Patterns that log matches are less specific yet. In all
cases the ability to override a pattern by matching another string, allows repetitive messages that
triggers the pattern, such as mailing lists, to pass the filter after the first one is processed manually.
The −s option allows deleted messages to be salvaged by either manual or semi-automatic review,
supporting the specification of more aggressive patterns. Finally, the utility of the pattern matcher
is not confined to filtering spam: it is a generally useful administrative tool for deleting inadver-
tently harmful messages, for example, mail loops, stuck senders or viruses. It is also useful for
collecting or counting messages matching certain criteria.

FILES
/mail/lib/patterns  default pattern file
/sys/log/smtpd     log of deleted messages
/mail/log/lines    file where log matches are logged
/mail/queue/*      directories where legitimate messages are queued for delivery
/mail/queue.dump/* directory where dumped messages are stored when the −s command
                    line option is specified.
/mail/copy/*       directory where copies of all incoming messages are stored.

SOURCE
/sys/src/cmd/upas/scanmail

SEE ALSO
    mail(1), qer(8), smtpd(6)

BUGS
    Testscan does not report a match when the body of a message contains exactly one line.
NAME
  screenlock – disable access to a terminal

SYNOPSIS
  screenlock

DESCRIPTION
  *Screenlock* grabs the screen, keyboard, and mouse devices to disable access to the Plan 9 terminal
  on which it is run. The screen can be unlocked by typing the invoking user’s Plan 9 password and
  a newline.

FILES
  /lib/bunny.bit  the image displayed while the terminal is locked

SOURCE
  /sys/src/cmd/screenlock.c

BUGS
  Use of this program on communal terminals is anti-social.
NAME
scuzz – SCSI target control

SYNOPSIS
scuzz [-6eq][-m max-xfer][[-r] sddev]

DESCRIPTION
Scuzz is an interactive program for exercising raw SCSI devices. Its intended purpose is to investigate and manipulate odd devices without the effort of writing a special driver, such as shuffling the media around on an optical jukebox. It reads commands from standard input and applies them to a SCSI target (other devices accessed through the sd(3) interface, such as ATA(PI) devices, may also work). If sddev is given on the command line, an open (see below) is immediately applied to the target. On successful completion of a command, ok n is printed, where n is the number of bytes transferred to/from the target; the −q command line option suppresses the ok message.

The −6 forces the use of 6-byte SCSI commands rather than 10-byte ones. Some older devices require this, though scuzz attempts to adapt automatically. The −e makes scuzz more willing to retry I/O errors but less tolerant of other errors and implies −6. This option is often needed to read Exabyte 8mm tapes. The −m option sets the maximum I/O transfer size to max-xfer. Exabyte drives often require this to be 1024 or the exact tape block size and some 4mm drives require this to be the exact tape block size or larger.

Commands
help command
Help is rudimentary and prints a one line synopsis for the named command, or for all commands if no argument is given.

probe
Probe attempts an inquiry command on all SCSI units, and prints the result preceded by the name of those targets which respond.

The help and probe commands may be given at any time.

open [−r] sddev
Open must be given before any of the remaining commands will be accepted. Internally, unless the −r option is given, open issues ready then inquiry, followed by a device class-specific command to determine the logical block size of the target. Sddev is an sd(3) device directory like /dev/sdC0.

close
Close need only be given if another target is to be opened in the current session.

The remaining commands are in rough groups, intended for specific classes of device. With the exception of the read, write, and space commands, all arguments are in the style of ANSI-C integer constants.

ready
Test Unit Ready checks if the unit is powered up and ready to do read and write commands.

rezero
Rezero Unit requests that a disk be brought to a known state, usually by seeking to track zero.

rewind
Rewind positions a tape at the beginning of current partition (there is usually only one partition, the beginning of tape).

reqsense
Request Sense retrieves Sense Data concerning an error or other condition and is usually issued following the completion of a command that had check-condition status. Scuzz automatically issues a reqsense in response to a check-condition status and prints the result.

format
Format Unit performs a “low level” format of a disk.

rblimits
Read Block Limits reports the possible block lengths for the logical unit. Tapes only.

read file nbytes
Read transfers data from the target to the host. A missing nbytes causes the entire device to be read.
write file nbytes
Write transfers data from the host to the target. A missing nbytes causes the entire input file to be transferred.

The first argument to the read and write commands specifies a source (write) or destination (read) for the I/O. The argument is either a plain file name or | followed by a command to be executed by rc(1). The argument may be quoted in the style of rc(1).

seek offset whence
Seek requests the target to seek to a position on a disk, arguments being in the style of seek(2); whence is 0 by default.

Scuzz maintains an internal notion of where the current target is positioned. The seek, read, write, rewind, rezero, and wtrack commands all manipulate the internal offset.

filemark howmany
Write Filemarks writes one (default) or more filemarks on a tape.

space [-b] [-f] [---howmany]
Space positions a tape forwards or backwards. The arguments specify logical block (-b) or filemark (-f) spacing; default is -b. If howmany is negative it specifies spacing backwards, and should be preceded by -- to turn off any further option processing. Default is 1.

inquiry Inquiry is issued to determine the device type of a particular target, and to determine some basic information about the implemented options and the product name.

modeselect bytes...
modeselect6 bytes...
Mode Select is issued to set variable parameters in the target. Bytes given as arguments comprise all the data for the target; see an appropriate manual for the format. The default is the 10-byte form of the command; modeselect6 is the 6-byte version.

modesense [page[nbytes]]
modesense6 [page[nbytes]]
Mode Sense reports variable and fixed parameters from the target. If no page is given, all pages are returned. Nbytes specifies how many bytes should be returned. The default is the 10-byte form of the command; modesense6 is the 6-byte version.

start [code]
stop [code]
eject [code]
ingest [code]
Start, stop, eject, and ingest are synonyms for Start/Stop Unit with different default values of code. Start/Stop Unit is typically used to spin up and spin down a rotating disk drive. Code is 0 to stop, 1 to start and 3 to eject (if the device supports ejection of the medium).

capacity Read Capacity reports the number of blocks and the block size of a disk.
The following commands are specific to CD and CD-R/RW devices. A brief description of each is given; see the SCSI-3 Multimedia Commands (MMC) Specification for details of arguments and interpretation of the results.

blank [track/LBA[type]]
Erase a CD-RW disk. Type identifies the method and coverage of the blanking.

rtoc [track/session-number[ses]]
The Read TOC/PMA command transfers data from one of the tables of contents (TOC or PMA) on the CD medium.

rdiscinfo
(Note the spelling.) Provides information about disks, including incomplete CD–
R/RW.

rtrackinfo [track]
Provides information about a track, regardless of its status.

cdpause
cdresume Pause/resume playback.
cdstop Stop playback.
cdplay [track-number] or [−r[LBA[length]]]
Play audio. With no arguments, starts at the beginning of the medium. If a track
number is given, the table of contents is read to find the playback start point. If the
−r option is given, block addressing is used to find the playback start point.

cdload [slot]
cdunload [slot]
Load/unload a disk from a changer.
cdstatus Read the mechanism status.

The following commands are specific to Media Changer devices. A brief description of each is
given; see the SCSI−3 Medium Changer Commands (SMC) Specification for details of arguments.
einit Initialize element status.
estatus type [length]
Report the status of the internal elements. Type 0 reports all element types.

mmmove transport source destination [invert]
Move medium.

FILES
/dev/sdXX/raw raw SCSI interface for command, I/O, and status.

SOURCE
/sys/src/cmd/scuzz

SEE ALSO
sd(3)
Small Computer System Interface − 2 (X3T9.2/86−109), .I Global Engineering Documents
SCSI Bench Reference, ENDL Publications
SCSI−3 Medium Changer Commands (SMC) Specification, .I www.t10.org

BUGS

Only a limited subset of SCSI commands has been implemented (as needed).
Only one target can be open at a time.
LUNs other than 0 are not supported.
No way to force 10−byte commands, though they are the default.
Should be recoded to use scsi(2) in order to get more complete sense code descriptions.
Scuzz betrays its origins by spelling rdiscinfo with a c even though the devices it manipulates
are spelled with a k.
The max−xfer value is currently limited to 245760 to limit kernel memory consumption.
It may be necessary to set max−xfer to exactly the block size used to write a tape in order to read
it on some drives.
NAME
secstored, secuser – secstore commands

SYNOPSIS
auth/secstored [-R] [-S servername] [-s address] [-x network] [-v]
auth/secuser [-v] username

DESCRIPTION
Secstored serves requests from secstore(1). By default it listens on port tcp!*15356; the -s option specifies an alternative address. In the connection protocol, secstored describes itself as service secstore, but the -S option can specify a different servername. The -R option supplements the password check with a call to a RADIUS server, for checking hardware tokens or other validation. The -x option specifies an alternative network to the default /net. By default, secstored puts itself into the background; the -v option enables a verbose debugging mode that suppresses that.

Secuser is an administrative command that runs on the secstore machine, normally the authserver, to create new accounts and to change status on existing accounts. It prompts for account information such as password and expiration date, writing to /adm/secstore/who/user for a given secstore user. The directory /adm/secstore should be created mode 770 with owner or group allowing access to the user that runs secstored. The -v option makes the command chattier.

By default, secstored warns the client if no account exists. If you prefer to obscure this information, use secuser to create an account FICTITIOUS.

FILES
/adm/secstore/who/user secstore account name, expiration date, verifier
/adm/secstore/store/user/ user’s file storage
/lib/ndb/auth for mapping local userid to RADIUS userid
/sys/log/secstore log file (if it does not exist, secstored logs to /dev/cons)

SOURCE
/sys/src/cmd/auth/secstore

SEE ALSO
secstore(1)
NAME
securenet – Digital Pathways SecureNet Key remote authentication box

DESCRIPTION
The SecureNet box is used to authenticate connections to Plan 9 from a foreign system such as a Unix machine or plain terminal. The box, which looks like a calculator, performs DES encryption with a key held in its memory. Another copy of the key is kept on the authentication server. Each box is protected from unauthorized use by a four digit PIN.

When the system requires SecureNet authentication, it prompts with a numerical challenge. The response is compared to one generated with the key stored on the authentication server. Respond as follows:

Turn on the box and enter your PIN at the EP prompt, followed by the ENT button. Enter the challenge at Ed prompt, again followed ENT. Then type to Plan 9 the response generated by the box. If you make a mistake at any time, reset the box by pressing ON. The authentication server compares the response generated by the box to one computed internally. If they match, the user is accepted.

The box will lose its memory if given the wrong PIN five times in succession or if its batteries are removed.

To reprogram it, type a 4 at the E0 prompt.

At the E1 prompt, enter your key, which consists of eight three-digit octal numbers. While you are entering these digits, the box displays a number ranging from 1 to 8 on the left side of the display. This number corresponds to the octal number you are entering, and changes when you enter the first digit of the next number.

When you are done entering your key, press ENT twice.

At the E2 prompt, enter a PIN for the box.

After you confirm by retyping the PIN at the E3 prompt, you can use the box as normal.

You can change the PIN using the following procedure. First, turn on the box and enter your current PIN at the EP prompt. Press ENT three times; this will return you to the EP prompt. Enter your PIN again, followed by ENT; you should see a Ed prompt with a – on the right side of the display. Enter a 0 and press ENT. You should see the E2 prompt; follow the instructions above for entering a PIN.

The SecureNet box performs the same encryption as the netcrypt routine (see encrypt(2)). The entered challenge, a decimal number between 0 and 100000, is treated as a text string with trailing binary zero fill to 8 bytes. These 8 bytes are encrypted with the DES algorithm. The first four bytes are printed on the display as hexadecimal numbers. However, when set up as described, the box does not print hexadecimal digits greater than 9. Instead, it prints a 2 for an A, B, or C, and a 3 for a D, E, or F. If a 5 rather than a 4 is entered at the E0 print, the hexadecimal digits are printed. This is not recommended, as letters are too easily confused with digits on the SecureNet display.

SEE ALSO
encrypt(2), auth(2)
Digital Pathways, Mountain View, California

BUGS
The box is clumsy to use and too delicate. If carried in a pocket, it can turn itself on and wear out the batteries.
NAME
  send - mail routing and delivery

SYNOPSIS
  upas/send [-b][-i][-r][-x][-#][ mailaddr ...]

DESCRIPTION
  Send is not normally run directly by the user. Instead, mail protocol agents like smtpd (see smtp(8)) and mail preparers like marshal(1) fork and execute send.

  Send reads a message from standard input and disposes of it in one of four ways:
  
  • If mailaddr refers to a local mailbox, it appends it to the recipient's mailbox.
  
  • If mailaddr is remote, it queues the mail for remote delivery.
  
  • If the -r option is given and the mail is undeliverable, it returns the mail to the sender.
  
  • if the -r option is not given and the mail is undeliverable, it appends the mail to
    /mail/box/username/dead.letter and prints a message to standard error.

  The file /mail/lib/rewrite determines exactly how to deliver or queue the mail. The decision is based purely on the recipient address.

  The options are:
  
  -b suppresses the addition of the To: line.
  -i let the message input be terminated by a line containing only a period, for compatibility with old mailers.
  -x do not send mail, but instead report the full mail address of the recipient.
  -# do not send mail, but instead report what command would be used to send the mail.
  -r input is via a pipe from another program. Expect a From line at the start of the message to provide the name of the sender and timestamp. This implies the -b option.

  Send uses the login name as the reply address.

FILES
  /sys/log/mail     mail log file
  /mail/box/*/dead.letter unmailable text
  /mail/lib/rewrite rules for handling addresses
  /mail/box/*/names  personal alias files
  /mail/lib/namefiles lists names of files containing system aliases

SOURCE
  /sys/src/cmd/upas/send

SEE ALSO
  aliasmail(8), faces(1), filter(1), mail(1), marshal(1), mlmgr(1), nedmail(1), qer(8), rewrite(6), smtp(8), upasfs(4)
NAME
smtp, smtpd - mail transport

SYNOPSIS
upas/smtp [-aAdfips] [-b busted-mx] ... [-g gateway] [-h host] [-u user] [-d domain]
    destaddr sender rcpt-list

upas/smtpd [-adDfg] [-c certfile] [-h mydom] [-k evilipaddr] [-m mailer] [-n netdir]

DESCRIPTION
Smtp sends the mail message from standard input to the users rcpt-list on the host at network
address address using the Simple Mail Transfer Protocol. The options are:

-a if the server supports PLAIN or LOGIN authentication, authenticate to the server using a pass-
word from factotum(4). See RFCs 3207 and 2554. This option implies -s.

-A autnic server: don’t wait for an SMTP greeting banner but immediately send a NOOP com-
mand to provoke the server into responding.

-b ignore busted-mx when trying MX hosts. May be repeated.

d turn on debugging to standard error.

-f just filter the converted message to standard output rather than sending it.

-g makes gateway the system to pass the message to if smtp can’t find an address nor MX entry
for the destination system.

-h use host as the local system name; it may be fully-qualified or not. If not specified, it will
default to the contents of /dev/sysname.

-i under -a, authenticate even if we can’t start TLS.

-p ping: just verify that the users named in the rcpt-list are valid users at destaddr; don’t send
any mail.

-s if the server supports the ESMTPO extension to use TLS encryption, turn it on for this session.
See RFC3207 for details.

-u specify a user name to be used in authentication. The default name is the current login id.

Finally if .domain is given, it is appended to the end of any unqualified system names in the enve-
lope or header.

Smtpd receives a message using the Simple Mail Transfer Protocol. Standard input and output are
the protocol connection. SMTP authentication by login and cram-md5 protocols is supported;
authenticated connections are permitted to relay.

The options are:

-a requires that all clients authenticate to be able to send mail.

-c specifies a certificate to use for TLS. Without this option, the capability to start TLS will not be
advertised.

-d turns on debugging output, with each connection’s output going to a uniquely-named file in
/sys/log/smtpd.

-D sleeps for 15 seconds usually at the start of the SMTP dialogue; this deters some spammers.
Connections from Class A networks frequented by spammers will incur a longer delay.

-f prevents relaying from non-trusted networks. It also tags messages from non-trusted sites
when they deliver mail from an address in a domain we believe we represent.

-g turns on grey/white list processing. All mail is rejected (with a retry code) unless the sender’s
IP address is on the whitelist, /mail/grey/whitelist, an append only file. Addresses
can be added to the whitelist by the administrator. However, the usual way for addresses to
be added is by smtpd itself. Whenever a message is received and the sender’s address isn’t
on the whitelist, smtpd first looks for the file /mail/grey/tmp/local/remote/recipient,
where local and remote are IP addresses of the local and remote systems, respectively. If it
exists and was created more than a few minutes go, the remote address is added to the white-list. If not, the file is created and the mail is rejected with a ‘try again’ code. The expectation is that spammers will not retry for more than a few minutes and that others will.

- h specifies the receiving domain. If this flag is not specified, the receiving domain is inferred from the host name.

- k causes connections from the host at the IP address, evilipaddr, to be dropped at program startup. Multiple addresses can be specified with several –k options. This option should be used carefully; it is intended to lessen the effects of denial of service attacks or broken mailers which continually connect. The connections are not logged and the remote system is not notified via the protocol.

- m set the mailer to which smtpd passes a received message. The default is /bin/upas/send.

- n specifies the name of the network directory assigned to the incoming connection. This is used to determine the peer IP address. If this flag is not specified, the peer address is determined using standard input.

- p permits clients to authenticate using protocols which transfer the password in the clear, e.g. login protocol. This should only be used if the connection has previously encrypted using e.g. tlssrv(8).

- r turns on forward DNS validation of non-trusted sender address.

- s causes copies of blocked messages to be saved in a sub-directory of /mail/queue.dump.

Smtpd is normally run by a network listener such as listen(8). Most of the command line options are more conveniently specified in the smtpd configuration file stored in /mail/lib/smtpd.conf.

SOURCE
/sys/src/cmd/upas/smtp

SEE ALSO
aliasmail(8), faces(1), filter(1), mail(1), marshal(1), mlmgr(1), nedmail(1), qer(8), rewrite(6), send(8), tlssrv(8), upasfs(4)
NAME
snoopy - spy on network packets

SYNOPSIS
snoopy [-CDdpst] [-M m] [-N n] [-F filter-expression] [-h first-header] [packet-source]
snoopy -? [proto...]

DESCRIPTION
Snoopy reads packets from a packet-source (default /net/ether0), matches them to a filter (by
default anything matches), and writes matching packets to standard output either in human readable form (default) or in a binary trace format that can be later read by snoopy. Packet-source can
be the name of an Ethernet (e.g., /net/ether0), an interface (e.g., /net/ipifc/0), or a file
of captured packets.

The human readable format consists of multiple lines per packet. The first line contains the mil-
seconds since the trace was started. Subsequent ones are indented with a tab and each contains
the dump of a single protocol header. The last line contains the dump of any contained data. For
example, a BOOTP packet would look like:

324389 ms
ether(s=0000929b1b54 d=ffffffffffff pr=0800 ln=342)
  ip(s=135.104.9.62 d=255.255.255.255 id=5099 frag=0000...)
  udp(s=68 d=67 ck=d151 ln= 308)
  bootp(t=Req ht=1 hl=16 hp=0 xid=217e5f27 sec=0 fl=800...)
  dhcp(t=Request clientid=0152415320704e7266238ebf01030...)

The binary format consists of:

2 bytes of packet length, msb first
8 bytes of nanosecond time, msb first
the packet

Filters are expressions specifying protocols to be traced and specific values for fields in the proto-
ocol headers. The grammar is:

expr: protocol
    | field '==' value
    | field '!=' value
    | protocol '(' expr ')' 
    | '(' expr ')' 
    | expr '||' expr
    | expr '&&' expr
    | '!' expr

The values for protocol and field can be obtained using the -? option. With no arguments, it lists
the known protocols. Otherwise it prints, for each protocol specified, which subprotocols it can
multiplex to, and which fields can be used for filtering. For example, the listing for ethernet is cur-
rently:

ether's filter attributes:
s  - source address
d  - destination address
a  - source|destination address
sd - source|destination address
t  - type
ether's subprotos:
0x0800 ip  0x8863 pppoe_disc
0x0806 arp  0x8864 pppoe_sess
0x0806 rarp  0x888e eapol
0x86dd ip6
The format of value depends on context. In general, ethernet addresses are entered as a string of hex digits; IP numbers in the canonical '.' format for v4 and ':' format for v6; and ports in decimal.

Snoopy's options are:

-\( -C \) compute the correct checksum for each packet; on mismatch, add a field \( !ck=xxxx \) where xxxx is the correct checksum.
-\( -D \) output will be a binary trace file in Unix pcap format.
-\( -d \) output will be a binary trace file.
-\( -t \) input is a binary trace file as generated with the \( -d \) option.
-\( -p \) do not enter promiscuous mode. Only packets to this interface will be seen.
-\( -s \) force one output line per packet. The default is multiline.
-\( -M \) discard all but the first \( m \) bytes of each packet. The default is to keep the entire packet. This option is most useful when writing packets to a file with the \( -d \) option.
-\( -N \) dump \( n \) data bytes per packet. The default is 32.
-\( -f \) use filter-expression to filter the packet stream. The default is to match all packets.
-\( -h \) assume the first header per packet to be of the first-header protocol. The default is ether.

EXAMPLES

To display only BOOTP and ARP packets:

\[
% \text{snoopy } -f '\text{arp } || \text{ bootp}'
\]

after optimize: ether(arp || ip(udp/bootp))

The first line of output shows the completed filter expression. Snoopy will fill in other protocols as necessary to complete the filter and then optimize to remove redundant comparisons.

To save all packets between 135.104.9.2 to 135.104.9.6 and later display those to/from TCP port 80:

\[
% \text{ramfs}
\]

\[
% \text{snoopy } -df 'ip(s=135.104.9.2 && d=135.104.9.6) || \text{ip(s=135.104.9.6 && d=135.104.9.2) > /tmp/quux}'
\]

<interrupt from the keyboard>

\[
% \text{snoopy } -tf 'tcp(sd=80)' /tmp/quux
\]

FILES

/net/ether0
  Ethernet device

SOURCE

/sys/src/cmd/ip/snoopy

BUGS

Snoopy only dumps ethernet packets, because there's no device to get IP packets without a media header.
NAME
stats - display graphs of system activity

SYNOPSIS
stats [−option] [ machine ... ]

DESCRIPTION
Stats displays a rolling graph of various statistics collected by the operating system and updated once per second. The statistics may be from a remote machine or multiple machines, whose graphs will appear in adjacent columns. The columns are labeled by the machine names and the number of processors on the machine if it is a multiprocessor.

The right mouse button presents a menu to enable and disable the display of various statistics; by default, stats begins by showing the load average on the executing machine.

The lower–case options choose the initial set to display:

- b  battery  percentage battery life remaining.
- c  context  number of process context switches per second.
- e  etherin  total number of packets sent and received per second.
- E  etherin, out  number of packets sent and received per second, displayed as separate graphs.
- f  fault  number of page faults per second.
- i  intr  number of interrupts per second.
- l  idle  system load, % time in idle, and % time in interrupts. The last two are averaged over all processors on a multiprocessor.
- L  load  (default) system load average. The load is computed as a running average of the number of processes ready to run, multiplied by 1000.
- m  mem  total pages of active memory. The graph displays the fraction of the machine’s total memory in use.
- n  etherin, out, err  number of packets sent and received per second, and total number of errors, displayed as separate graphs.
- p  tlbpurge  number of translation lookaside buffer flushes per second.
- s  syscall  number of system calls per second.
- t  tlbmiss  number of translation lookaside buffer misses per second.
- w  swap  number of valid pages on the swap device. The swap is displayed as a fraction of the number of swap pages configured by the machine.
- S  802.11b  display the signal strength detected by the 802.11b wireless ether card; the value is usually below 50% unless the receiver is in the same room as the transmitter, so a midrange value represents a strong signal.

The graphs are plotted with time on the horizontal axis. The vertical axes range from 0 to 1000*sleepsecs, multiplied by the number of processors on the machine when appropriate. The only exceptions are memory, and swap space, which display fractions of the total available, system load, which displays a number between 0 and 1000, idle and intr, which display percentages and the Ethernet error count, which goes from 0 to 10. If the value of the parameter is too large for the visible range, its value is shown in decimal in the upper left corner of the graph.

Upper–case options control details of the display. All graphs are affected; there is no mechanism to affect only one graph.

−T  sleepsecs
Set the number of seconds between samples to sleepsecs (default one second). Sleepsecs may be a floating–point number.

−S  scale
Sets a scale factor for the displays. A value of 2, for example, means that the highest value plotted will be twice as large as the default.

−L  Plot all graphs with logarithmic y axes. The graph is plotted so the maximum value that would be displayed on a linear graph is 2/3 of the way up the y axis and the total range of the graph is a factor of 1000; thus the y origin is 1/100 of the default maximum value and the top of the graph is 10 times the default maximum.
−Y If the display is large enough to show them, place value markers along the y axes of the graphs. Since one set of markers serves for all machines across the display, the values in the markers disregard scaling factors due to multiple processors on the machines. On a graph for a multiprocessor, the displayed values will be larger than the markers indicate. The markers appear along the right, and the markers show values appropriate to the right-most machine; this only matters for graphs such as memory that have machine-specific maxima.

FILES
/net/ether0/0/stats
#c/swap
#c/sysstat

SOURCE
/sys/src/cmd/stats.c

BUGS
Some machines do not have TLB hardware.
NAME
  statusbar - display a bar graph status window

SYNOPSIS
  aux/statusbar [-kt] [-w minx, miny, maxx, maxy] title

DESCRIPTION
  Aux/statusbar reads textual status lines from standard input, converting them into a continuously updated bar graph displayed in a new window on the screen. The title is displayed on a line above the bar graph. Each input line is two space-separated decimal numbers: the numerator and denominator of a fraction.

  Statusbar exits when it reaches end-of-file on standard input. Typing DEL or control-C will also cause it to exit.

  The options are:
  -k  do not allow typing to cause statusbar to exit
  -t  print an ASCII status bar to standard output, using backspace to redraw it
  -w  set the coordinates of the statusbar window created

EXAMPLE
  The -v option to hget(1) causes it to print status lines suitable for input to statusbar.
  Monitor a long download:
      aux/statusbar 'big file download'

SOURCE
  /sys/src/cmd/aux/statusbar.c
NAME
stub - provide mount point stubs

SYNOPSIS
aux/stub [ −Dd ] path/name

DESCRIPTION
Aux/stub union mounts itself before path in the name space. It serves a file system containing a single entry, name, with file mode 0. The intent is to provide a place to bind or mount other resources. The options are:

−D print all 9P messages
−d make name a directory; by default it is a file

EXAMPLE
Use stub and sshnet (see ssh(1)) to create a new network protocol “mit” that is actually TCP tunneled via SSH to a machine at MIT:

% sshnet −m /net.alt amsterdam.lcs.mit.edu
% aux/stub −d /net/mit
% bind /net.alt/tcp /net/mit
% con −l mit!plan9.bell-labs.com!whoami
connected to mit!plan9.bell-labs.com!whoami on /net/mit/0
i am 204.178.31.2 sysname achille you are 18.26.4.9 port 1248
%

SOURCE
/sys/src/cmd/aux/stub.c

SEE ALSO
mntgen(4)
NAME
swap - establish a swap file

SYNOPSIS
swap file

DESCRIPTION
Swap establishes a file or device for the system to swap on. If file is a device, the device is used directly; if a directory, a unique file is created in that directory on which to swap. The environment variable swap is set to the full name of the resulting file. The number of blocks available in the file or device must be at least the number of swap blocks configured at system boot time.

If a swap channel has already been set and no blocks are currently valid in the file the old file will be closed and then replaced. If any blocks are valid on the device an error is returned instead.

SOURCE
/sys/src/cmd/swap.c

BUGS
Swapping to a file served by a local user-level process, such as kfs(4), can lead to deadlock; use raw devices or remote files instead.
NAME

timesync - synchronize the system clock to a time source

SYNOPSIS

aux(timesync [-a accuracy] [-S stratum] [-s netroot] [-frnDdLiLg] [timeserver]

DESCRIPTION

Aux/timesync synchronizes the system clock to a time source, by default a file server. The options are:

- f synchronize to a file server. If timeserver is missing, use /srv/boot.
- r synchronize to the local real time clock, #r/rtc.
- L used with -r to indicate the real time clock is in local time rather than GMT. This is useful on PCs that also run the Windows OS.
- n synchronize to an NTP server. If timeserver is missing, dial the server udp!$ntp!ntp.
- D print debugging to standard error
- d put file containing last determined clock frequency in directory dir, default /tmp.
- i stands for impotent. Timesync announces what it would do but doesn’t do it. This is useful for tracking alternate time sources.
- a specifies the accuracy in nanoseconds to which the clock should be synchronized. This determines how often the reference clock is accessed.
- G causes timesync to use a gps server (see gpsfs(8)) as a time source.
- s causes timesync to listen for UDP NTP requests on the network rooted at netroot. Up to 4 -s options are allowed.
- S sets the stratum number to stratum.
- l turns on logging to /sys/log/timesync.

FILES

/tmp/ts.<sysname>.<type>.timeserver where the last frequency guess is kept
/sys/log/timesync log file

SOURCE

/sys/src/cmd/aux/timesync.c
NAME
tlssrv, tlscient, tlssrvtunnel, tlsclienttunnel - TLS server and client

SYNOPSIS
tlssrv [-c cert.pem][-l logfile][-r remotesys] cmd [args ...]
tlscient [-t trustedkeys] [-x excludedkeys] address
tlssrvtunnel plain-addr crypt-addr cert.pem
     tlsclienttunnel crypt-addr plain-addr trustedkeys

DESCRIPTION
Tlssrv is a helper program, typically exec'd in a /bin/service file to establish an SSL or TLS
connection before launching cmd args; a typical command might start the IMAP or HTTP server.
Cert.pem is the server certificate; factotum(4) should hold the corresponding private key. The
specified logfile is by convention the same as for the target server. Remotesys is mainly used for
logging.

Tlscient is the reverse of tlssrv: it dials address, starts TLS, and then relays between the network
connection and standard input and output. If the -t flag (and, optionally, the -x flag) is given,
the remote server must present a key whose SHA1 hash is listed in the file trustedkeys but not in
the file excludedkeys. See thumbprint(6) for more information.

Tlssrvtunnel and tlsclienttunnel use these tools and listen(8) (see listen(8)) to provide TLS network
tunnels, allowing legacy application to take advantage of TLS encryption.

EXAMPLES
Listen for TLS-encrypted IMAP by creating a server certificate /sys/lib/tls/imap.pem and a
listener script /bin/service.auth/tcp993 containing:
    #!/bin/rc
    exec tlssrv -c /sys/lib/tls/imap.pem -limap4d -r '{cat $3/remote} 
      /bin/ip/imap4d -p -dyourdomain -r '{cat $3/remote} 
      >[2]/sys/log/imap4d

Interact with the server, putting the appropriate hash into /sys/lib/tls/mail and running:
    tlsclient -t /sys/lib/tls/mail tcp!server!imaps

Create a TLS-encrypted VNC connection from a client on kremvax to a server on moscvax:
    mosc% vncs -d :3
    mosc% tlssrvtunnel tcp!moscvax!5903 tcp!*!12345 
      /usr/you/lib/cert.pem
    krem% tlsclienttunnel tcp!moscvax!12345 tcp!*!5905 
      /usr/you/lib/cert.thumb
    krem% vncv kremvax:5

(The port numbers passed to the VNC tools are offset by 5900 from the actual TCP port numbers.)

FILES
    /sys/lib/tls

SOURCE
    /sys/src/cmd/tlssrv.c
    /sys/src/cmd/tlsclient.c
    /rc/bin/tlssrvtunnel
    /rc/bin/tlsclienttunnel

SEE ALSO
    factotum(4), listen(8), rsa(8)
    Unix's stunnel
NAME
    trampoline - forward incoming calls to another address

SYNOPSIS
    aux/trampoline [-9] [-a altaddr] [-m netdir] addr

DESCRIPTION
    *Trampoline* can be used in a service file (see *listen*(8)) to link an incoming call to another address
    that provides the service, typically on another machine.

    *Trampoline* dials *addr* and copies data between that connection and its own standard input and
    output.

    The options are:
    -9 The connection carries only 9P messages. In this case *trampoline* will relay whole mes-
      sages at a time.

    -a altaddr
        Dial *altaddr* and relay between the two network connections, ignoring standard input and
        output.

    -m netdir
        Restrict forwarding to particular machines. *Netdir* must be the incoming call directory. *Trampoline*
        finds the caller’s MAC address *m* and checks that *ndb*(6) contains an entry with
        *ether*= *m* and the attribute *trampok*. If no such entry is found, the call is rejected.

FILES
    /sys/log/trampoline  logs rejected calls

SOURCE
    /sys/src/cmd/aux/trampoline.c

SEE ALSO
    *dial*(2), *listen*(8)
NAME
  udpecho - echo UDP packets

SYNOPSIS
  ip/udpecho [-x ext]

DESCRIPTION
  Listen on UDP port 7 and echo back any packets received. This should only be run for testing since it can be used to disguise the identity of someone doing a denial of service attack.
NAME
bootfloppy, bootplan9, bootwin9x, bootwinnt, personalize, setup.9fat, setup.disk, setup.kfs, update - administration for local file systems

SYNOPSIS
pc/bootfloppy floppydisk plan9.ini
pc/bootplan9 /dev/sdXX
pc/bootwin9x
pc/bootwinnt
pc/personalize
pc/setup.9fat /dev/sdXX/9fat plan9.ini
pc/setup.disk /dev/sdXX plan9.ini
pc/update

DESCRIPTION
These programs help maintain a file system on a local disk for a private machine.
Setup.disk partitions a disk and makes a new file system on the disk. It then calls setup.9fat, update, and personalize to initialize the file system.
Setup.9fat formats the named 9fat partition, installing /386/9load, /386/9pcdisk, and the named plan9.ini file.
Update copies the current kernel to the disk and updates files on the local file system by copying them from the main file server (named by the environment variable $fileservers). The files it updates are specified by the mkfs(8) prototype file /sys/lib/sysconfig/proto/386proto.
Personalize removes the contents of the /usr directory on the local disk and copies a minimal set of files for the user who runs the command.
The boot scripts prepare various ways to bootstrap Plan 9. Bootfloppy creates a boot floppy containing 9load, a zeroed 512-byte plan9.nvr, and the named file as plan9.ini. Bootplan9 sets the 9fat partition to be the active partition, the one used at boot time. Bootwin9x edits the files config.sys, msdos.sys, and autoexec.bat on the drive mounted by c: to provide Plan 9 as a boot menu option. These system files are first backed up as config.p9, msdos.p9, and autoexec.p9. Bootwinnt edits the Windows NT boot loader menu contained in the first FAT partition's boot.ini to provide Plan 9 as an option. It is first backed up as boot.p9. If backup files already exist, bootwin9x and bootwinnt do nothing.

FILES
/sys/lib/sysconfig/proto/ Mkfs(8) prototype files.

SOURCE
/rc/bin/pc/*

SEE ALSO
kfs(4), 9load(8), mkfs(8), prep(8), sd(3)
“Installing the Plan 9 Distribution”.

386/9pcdisk

NAME
venti - archival storage server

SYNOPSIS
venti/venti [-Ld] [-a address] [-B blockcachesize] [-c config] [-C lumpcachesize]
[-h httpaddress] [-I indexcachesize] [-m free-memory%] [-W webroot]

DESCRIPTION
Venti is a SHA1-addressed archival storage server. See venti(6) for a full introduction to the system. This page documents the structure and operation of the server.

A venti server requires multiple disks or disk partitions, each of which must be properly formatted before the server can be run.

Disk
The venti server maintains three disk structures, typically stored on raw disk partitions: the append-only data log, which holds, in sequential order, the contents of every block written to the server; the index, which helps locate a block in the data log given its score; and optionally the bloom filter, a concise summary of which scores are present in the index. The data log is the primary storage. To improve the robustness, it should be stored on a device that provides RAID functionality. The index and the bloom filter are optimizations employed to access the data log efficiently and can be rebuilt if lost or damaged.

The data log is logically split into sections called arenas, typically sized for easy offline backup (e.g., 500MB). A data log may comprise many disks, each storing one or more arenas. Such disks are called arena partitions. Arena partitions are filled in the order given in the configuration.

The index is logically split into block-sized pieces called buckets, each of which is responsible for a particular range of scores. An index may be split across many disks, each storing many buckets. Such disks are called index sections.

The index must be sized so that no bucket is full. When a bucket fills, the server must be shut down and the index made larger. Since scores appear random, each bucket will contain approximately the same number of entries. Index entries are 40 bytes long. Assuming that a typical block being written to the server is 8192 bytes and compresses to 4096 bytes, the active index is expected to be about 1% of the active data log. Storing smaller blocks increases the relative index footprint; storing larger blocks decreases it. To allow variation in both block size and the random distribution of scores to buckets, the suggested index size is 5% of the active data log.

The (optional) bloom filter is a large bitmap that is stored on disk but also kept completely in memory while the venti server runs. It helps the venti server efficiently detect scores that are not already stored in the index. The bloom filter starts out zeroed. Each score recorded in the bloom filter is hashed to choose nhash bits to set in the bloom filter. A score is definitely not stored in the index of any of its nhash bits are not set. The bloom filter thus has two parameters: nhash (maximum 32) and the total bitmap size (maximum 512MB, 2^{32} bits).

The bloom filter should be sized so that \( nhash \times nblock \leq 0.7 \times b \), where nblock is the expected number of blocks stored on the server and b is the bitmap size in bits. The false positive rate of the bloom filter when sized this way is approximately \( 2^{-nblock} \). Nhash less than 10 are not very useful; nhash greater than 24 are probably a waste of memory. Fmtbloom (see venti–fmt(8)) can be given either nhash or nblock; if given nblock, it will derive an appropriate nhash.

Memory
Venti can make effective use of large amounts of memory for various caches.

The lump cache holds recently-accessed venti data blocks, which the server refers to as lumps. The lump cache should be at least 1MB but can profitably be much larger. The lump cache can be thought of as the level-1 cache: read requests handled by the lump cache can be served instantly.

The block cache holds recently-accessed disk blocks from the arena partitions. The block cache needs to be able to simultaneously hold two blocks from each arena plus four blocks for the currently-filling arena. The block cache can be thought of as the level-2 cache: read requests handled by the block cache are slower than those handled by the lump cache, since the lump data must be extracted from the raw disk blocks and possibly decompressed, but no disk accesses are necessary.
The **index cache** holds recently-accessed or prefetched index entries. The index cache needs to be able to hold index entries for three or four arenas, at least, in order for prefetching to work properly. Each index entry is 50 bytes. Assuming 500MB arenas of 128,000 blocks that are 4096 bytes each after compression, the minimum index cache size is about 6MB. The index cache can be thought of as the level–3 cache: read requests handled by the index cache must still go to disk to fetch the arena blocks, but the costly random access to the index is avoided.

The size of the index cache determines how long venti can sustain its ‘burst’ write throughput, during which time the only disk accesses on the critical path are sequential writes to the arena partitions. For example, if you want to be able to sustain 10MB/s for an hour, you need enough index cache to hold entries for 36GB of blocks. Assuming 8192-byte blocks, you need room for almost five million index entries. Since index entries are 50 bytes each, you need 250MB of index cache. If the background index update process can make a single pass through the index in an hour, which is possible, then you can sustain the 10MB/s indefinitely (at least until the arenas are all filled).

The **bloom filter** requires memory equal to its size on disk, as discussed above.

A reasonable starting allocation is to divide memory equally (in thirds) between the bloom filter, the index cache, and the lump and block caches; the third of memory allocated to the lump and block caches should be split unevenly, with more (say, two thirds) going to the block cache.

**Network**

The venti server announces two network services, one (conventionally TCP port venti, 17034) serving the venti protocol as described in venti(6), and one serving HTTP (conventionally TCP port http, 80).

The venti web server provides the following URLs for accessing status information:

/\index  A summary of the usage of the arenas and index sections.
/xindex  An XML version of /\index.
/storage Brief storage totals.
/set/variable

The current integer value of **variable**. Variables are: compress, whether or not to compress blocks (for debugging); logging, whether to write entries to the debugging logs; stats, whether to collect run-time statistics; icachesleeptime, the time in milliseconds between successive updates of megabytes of the index cache; arenasumsleeptime, the time in milliseconds between reads while checksumming an arena in the background. The two sleep times should be (but are not) managed by venti; they exist to provide more experience with their effects. The other variables exist only for debugging and performance measurement.

/set/variable/value
Set **variable** to **value**.

/graph/name/param/param
A PNG image graphing the named run-time statistic over time. The details of names and parameters are undocumented; see httpd.c in the venti sources.

/log  A list of all debugging logs present in the server’s memory.
/log/name  The contents of the debugging log with the given name.

flushcache  Force venti to begin flushing the index cache to disk. The request response will not be sent until the flush has completed.

flushdcache  Force venti to begin flushing the arena block cache to disk. The request response will not be sent until the flush has completed.

Requests for other files are served by consulting a directory named in the configuration file (see webrouter below).

**Configuration File**

A venti configuration file enumerates the various index sections and arenas that constitute a venti
system. The components are indicated by the name of the file, typically a disk partition, in which they reside. The configuration file is the only location that file names are used. Internally, venti uses the names assigned when the components were formatted with fmtarenas or fmtisect (see venti–fmt(8)). In particular, only the configuration needs to be changed if a component is moved to a different file.

The configuration file consists of lines in the form described below. Lines starting with # are comments.

index name Names the index for the system.
arenas file File is an arena partition, formatted using fmtarenas.
isect file File is an index section, formatted using fmtisect.
bloom file File is a bloom filter, formatted using fmtbloom.

After formatting a venti system using fmtindex, the order of arenas and index sections should not be changed. Additional arenas can be appended to the configuration; run fmtindex with the -a flag to update the index.

The configuration file also holds configuration parameters for the venti server itself. These are:

mem size lump cache size
bcmem size block cache size
icmem size index cache size
addr netaddr network address to announce venti service (default tcp:*!venti)
httpaddr netaddr network address to announce HTTP service (default tcp:*!http)
queue writes queue writes in memory (default is not to queue)
webroot dir directory tree containing files for venti’s internal HTTP server to consult for unrecognized URLs

The units for the various cache sizes above can be specified by appending a k, m, or g (case-insensitive) to indicate kilobytes, megabytes, or gigabytes respectively.

The file name in the configuration lines above can be of the form file:lo–hi to specify a range of the file. Lo and hi are specified in bytes but can have the usual k, m, or g suffixes. Either lo or hi may be omitted. This notation eliminates the need to partition raw disks on non–Plan 9 systems.

Command Line
Many of the options to Venti duplicate parameters that can be specified in the configuration file. The command line options override those found in a configuration file. Additional options are:

-c config The server configuration file (default venti.conf)
-d Produce various debugging information on standard error. Implies -s.
-L Enable logging. By default all logging is disabled. Logging slows server operation considerably.
-m Allocate free–memory% percent of the available free RAM, and partition it per the guidelines in the Memory subsection. This percentage should be large enough to include the entire bloom filter. This overrides all other memory sizing parameters, including those on the command line and in the configuration file.
-r Allow only read access to the venti data.
-s Do not run in the background. Normally, the foreground process will exit once the Venti server is initialized and ready for connections.

EXAMPLE
A simple configuration:
% cat venti.conf
index main
isect /tmp/disks/isect0
isect /tmp/disks/isect1
arenas /tmp/disks/arenas
bloom /tmp/disks/bloom
%
Format the index sections, the arena partition, the bloom filter, and finally the main index:

```bash
% venti/fmtisect isect0. /tmp/disks/isect0
% venti/fmtisect isect1. /tmp/disks/isect1
% venti/fmtarenas arenas0. /tmp/disks/arenas &
% venti/fmtbloom /tmp/disks/bloom &
% wait
% venti/fmtindex venti.conf
%
```

Start the server and check the storage statistics:

```bash
% venti/venti
% hget http://$sysname/storage
```

**SOURCE**

/sys/src/cmd/venti/srv

**SEE ALSO**

venti(1), venti(2), venti(6), venti–backup(8), venti–fmt(8)


**BUGS**

Setting up a venti server is too complicated.
NAME
rdarena, wrarena - copy arenas between venti servers

SYNOPSIS
venti/rdarena [-qv] arenapart arenaname
venti/wrarena [-o fileoffset] [-h host] arenafile [clumpoffset]

DESCRIPTION
Rdarena extracts the named arena from the arena partition arenapart and writes this arena to standard output. This command is typically used to back up an arena to external media. The -v option generates more verbose output on standard error; -q generates only errors on standard error.

Wrarena writes the blocks contained in the arena arenafile (typically, the output of rdarena) to a Venti server. It is typically used to reinitialize a Venti server from backups of the arenas. For example,

venti/rdarena /dev/sdC0/arenas arena.0 >external.media
venti/wrarena -h venti2 external.media

writes the blocks contained in arena.0 to the Venti server venti2 (typically not the one using /dev/sdC0/arenas).

The -o option specifies that the arena starts at byte fileoffset (default 0) in arenafile. This is useful for reading directly from the Venti arena partition:

venti/wrarena -h venti2 -o 335872 /dev/sdC0/arenas

(In this example, 335872 is the offset shown in the Venti server’s index list (344064) minus one block (8192). You will need to substitute your own arena offsets and block size.)

Finally, the optional offset argument specifies that the writing should begin with the clump starting at offset within the arena. Wrarena prints the offset it stopped at (because there were no more data blocks). This could be used to incrementally back up a Venti server to another Venti server:

last='{cat last}
venti/wrarena -h venti2 -o 335872 /dev/sdC0/arenas $last >output
awk '/^end offset/ { print $3 }' offset >last

Of course, one would need to add wrapper code to keep track of which arenas have been processed. See /sys/src/cmd/venti/words/backup.example for a version that does this.

SOURCE
/sys/src/cmd/venti/srv

SEE ALSO
venti(6), venti(8)

BUGS
Wrarena can’t read a pipe or network connection containing an arena; it needs a file already containing the entire arena.
NAME
buildindex, checkarenas, checkindex, conf, fmtarenas, fmtbloom, fmtindex, fmtisect, syncindex – prepare and maintain a venti server

SYNOPSIS
venti/fmtarenas [-Z] [-a arenasize] [-b blocksize] name file
venti/fmtisect [-1Z] [-b blocksize] name file
venti/fmtbloom [-n nbloeks] [-N nhash] [-s size] file
venti/fmtindex [-a] venti.conf
venti/conf [-w] partition [configfile]
venti/buildindex [-bd] [-i isect] ... [-M imemsize] venti.conf
venti/checkindex [-f] [-B blockcachesize] venti.conf tmp
venti/checkarenas [-afv] file

DESCRIPTION
These commands aid in the setup, maintenance, and debugging of venti servers. See venti(6) for an overview of the venti system and venti(8) for an overview of the data structures used by the venti server.

Note that the units for the various sizes in the following commands can be specified by appending k, m, or g to indicate kilobytes, megabytes, or gigabytes respectively.

Formatting
To prepare a server for its initial use, the arena partitions and the index sections must be formatted individually, with fmtarenas and fmtisect. Then the collection of index sections must be combined into a venti index with fmtindex.

Fmtarenas formats the given file, typically a disk partition, into an arena partition. The arenas in the partition are given names of the form name%d, where %d is replaced with a sequential number starting at 0.

Options to fmtarenas are:
- a arenasize
  The arenas are of arenasize bytes. The default is 512M, which was selected to provide a balance between the number of arenas and the ability to copy an arena to external media such as recordable CDs and tapes.
- b blocksize
  The size, in bytes, for read and write operations to the file. The size is recorded in the file, and is used by applications that access the arenas. The default is 8k.
- 4
  Create a ‘version 4’ arena partition for backwards compatibility with old servers. The default is version 5, used by the current venti server.
- Z
  Do not zero the data sections of the arenas. Using this option reduces the formatting time but should only be used when it is known that the file was already zeroed. (Version 4 only; version 5 sections are not and do not need to be zeroed.)

Fmtisect formats the given file, typically a disk partition, as a venti index section with the specified name. Each of the index sections in a venti configuration must have a unique name.

Options to fmtisect are:
- b bucketsize
  The size of an index bucket, in bytes. All the index sections within a index must have the same bucket size. The default is 8k.
- 1
  Create a ‘version 1’ index section for backwards compatibility with old servers. The default is version 2, used by the current venti server.
- Z
  Do not zero the index. Using this option reduces the formatting time but should only be used when it is known that the file was already zeroed. (Version 1 only; version 2 sections are not and do not need to be zeroed.)
Fmtbloom formats the given file as a Bloom filter (see venti(6)). The options are:

- `n nblock` | `-N nhash`
  The number of blocks expected to be indexed by the filter or the number of hash functions to use. If the `-n` option is given, it is used, along with the total size of the filter, to compute an appropriate `nhash`.

- `s size` The size of the Bloom filter. The default is the total size of the file. In either case, `size` is rounded down to a power of two.

The `file` argument in the commands above can be of the form `file:lo−hi` to specify a range of the file. `Lo` and `hi` are specified in bytes but can have the usual `k`, `m`, or `g` suffixes. Either `lo` or `hi` may be omitted. This notation eliminates the need to partition raw disks on non−Plan 9 systems.

Fmtindex reads the configuration file `venti.conf` and initializes the index sections to form a usable index structure. The arena files and index sections must have previously been formatted using `fmtarenas` and `fmtisect` respectively.

The function of a venti index is to map a SHA1 fingerprint to a location in the data section of one of the arenas. The index is composed of blocks, each of which contains the mapping for a fixed range of possible fingerprint values. Fmtindex determines the mapping between SHA1 values and the blocks of the collection of index sections. Once this mapping has been determined, it cannot be changed without rebuilding the index. The basic assumption in the current implementation is that the index structure is sufficiently empty that individual blocks of the index will rarely overflow. The total size of the index should be about 2% to 10% of the total size of the arenas, but the exact percentage depends both on the index block size and the compressed size of blocks stored. See the discussion in venti(8) for more.

Fmtindex also computes a mapping between a linear address space and the data section of the collection of arenas. The `−a` option can be used to add additional arenas to an index. To use this feature, add the new arenas to `venti.conf` after the existing arenas and then run `fmtindex −a`.

A copy of the above mappings is stored in the header for each of the index sections. These copies enable `buildindex` to restore a single index section without rebuilding the entire index.

To make it easier to bootstrap servers, the configuration file can be stored in otherwise empty space at the beginning of any venti partitions using `conf`. A partition so branded with a configuration file can be used in place of a configuration file when invoking any of the venti commands. By default, `conf` prints the configuration stored in `partition`. When invoked with the `−w` flag, `conf` reads a configuration file from `configfile` (or else standard input) and stores it in `partition`.

Checking and Rebuilding

Buildindex populates the index for the Venti system described in `venti.conf`. The index must have previously been formatted using `fmtindex`. This command is typically used to build a new index for a Venti system when the old index becomes too small, or to rebuild an index after media failure. Small errors in an index can usually be fixed with `checkindex`, but `checkindex` requires a large temporary workspace and `buildindex` does not.

Options to `buildindex` are:

- `b` Reinitialise the Bloom filter, if any.
- `d` ‘Dumb’ mode; run all three passes.
- `i isect` Only rebuild index section `isect`; may be repeated to rebuild multiple sections. The name `none` is special and just reads the arenas.
- `M imemsize` The amount of memory, in bytes, to use for caching raw disk accesses while running `buildindex`. (This is not a property of the created index.) The usual suffices apply. The default is 256M.

Checkindex examines the Venti index described in `venti.conf`. The program detects various error conditions including: blocks that are not indexed, index entries for blocks that do not exist, and duplicate index entries. If requested, an attempt can be made to fix errors that are found.

The `tmp` file, usually a disk partition, must be large enough to store a copy of the index. This temporary space is used to perform a merge sort of index entries generated by reading the arenas.
Options to checkindex are:
  -B blockcachesize
      The amount of memory, in bytes, to use for caching raw disk accesses while running checkindex. The default is 8k.
  -f    Attempt to fix any errors that are found.

Checkarenas examines the Venti arenas contained in the given file. The program detects various error conditions, and optionally attempts to fix any errors that are found.

Options to checkarenas are:
  -a    For each arena, scan the entire data section. If this option is omitted, only the end section of the arena is examined.
  -f    Attempt to fix any errors that are found.
  -v    Increase the verbosity of output.

SOURCE
/sys/src/cmd/venti/srv

SEE ALSO
venti(6), venti(8)

BUGS
Buildindex should allow an individual index section to be rebuilt.
NAME
vga – configure a VGA card

SYNOPSIS
aux/vga [-BcdeflpvV] [−b bios-string] [−m monitor] [−x file] [mode [size]]

DESCRIPTION
Vga configures a VGA controller for various display sizes and depths. Using the monitor type specified in /env/monitor (default vga) and the mode given as argument (default 640x480x1), vga uses the database of known VGA controllers and monitors in /lib/vgadb (see vgadb(6)) to configure the display via the devices provided by vga(3). The options are:
−b bios-string
use the VGA database entry corresponding to bios-string (e.g. 0xC0045="Stealth 64 DRAM Vers. 2.02") rather than looking for identifying strings in the BIOS memory.
−B
dump the BIOS memory (in hex) to standard output and exit.
−c
disable the use of the hardware graphics cursor.
−d
include the color palette in whatever actions are performed, usually printing the contents.
−i
when used with −p display the register values that will be loaded.
−l
load the desired mode.
−m monitor
override the /env/monitor value. /env/monitor is usually set by including it in the plan9.ini file read by the PC boot program 9load(8).
−p
print the current or expected register values at appropriate points depending on other options.
−v
print a trace of the functions called.
−V
print a verbose trace of the functions called.
−x file
use file as the VGA database rather than /lib/vgadb.

Mode is of the form XxYxZ, where X, Y, and Z are numbers specifying the display height, width, and depth respectively. The mode must appear in /lib/vgadb as a value for one of the monitor entries. The usual modes are 640x480x[18], 800x600x[18], 1024x768x[18][i], 1280x1024x[18][i], 1376x1024x8, and 1600x1200x8. A trailing i indicates interlaced operation. The default mode is 640x480x8. Size is of the form XxY and configures the display to have a virtual screen of the given size. The physical screen will pan to follow the mouse. This is useful on displays with small screens, such as laptops, but can be confusing.

Using the monitor name vesa instructs vga to use VESA BIOS calls to configure the display. Also, if our VGA controller can’t be found in vgadb, vga will try the VESA calls. There are no entries for the vesa monitor in vgadb. For a list of available VESA modes, use

aux/vga -m vesa -p

Loading the special mode text:
aux/vga -l text

switches out of graphics mode back into text mode. It uses the VESA BIOS.

EXAMPLES
Change the display resolution:
aux/vga -l 1600x1200x8

Print the current VGA controller registers. It is usually best to redirect the output of a −p command to a file to prevent confusion caused by using the VGA controller while trying to dump its state:

aux/vga −p >/tmp/x
Force the VGA controller to a known state:
   aux/vga -m vga -l
Print the current VGA controller state and what would be loaded into it for a new resolution, but
don’t do the load:
   aux/vga -ip 1376x1024x8 >/tmp/x

FILES
/env/monitor  display type (default vga).
/lib/vgadb     VGA configuration file.

SOURCE
/sys/src/cmd/aux/vga

SEE ALSO
   vga(3), vgadb(6), 9load(8)

BUGS
Aux/vga makes every effort possible to verify that the mode it is about to load is valid and will
bail out with an error message before setting any registers if it encounters a problem. However,
things can go wrong, especially when playing with a new VGA controller or monitor setting. It is
useful in such cases to have the above command for setting the controller to a known state at your
fingertips.
NAME
   wol - send wake-on-lan Ethernet packet

SYNOPSIS
   ip/wol [−v][−a dialstr][−c password] macaddr

DESCRIPTION
   Wol sends a magic wake-on-lan Ethernet packet to dialstr (default udp!255.255.255.255:0, the IPv4 broadcast address), intended to wake up the machine with an Ethernet interface with the MAC address macaddr. Macaddr is not used to route the packet, but is inserted into the magic packet as required by the wake-on-lan protocol.

   An optional password of at most six bytes can be sent. The option −v prints verbose information about the packet sent.

SEE ALSO
   dial(2), parseether in ip(2)
PERMUTED INDEX

Manual pages for all sections are accessible on line through man(1).
To save space, neighboring references to the same page have been collapsed into a single reference.
This should cause no difficulty in cases like ‘atan’ and ‘atan2’, but is somewhat obscure in the case of ‘strcat’ and ‘strchr’.

exec, exit, flag, forkl, shift, wait, whatis, ... 0a, 1a, 2a, 5a, 6a, 7a, 8a, qa, va - ....... 0c, 1c, 2c, 5c, 6c, 7c, 8c, kc, qc, vc - C ... 01, 11, 21, 51, 61, 71, 81, k1, ql, vl - loaders ... 2(1) 8
assemblers .......................... mp3dec(1) 138
compilers .......................... vt - emulate a 100 or VT-220 terminal ................................... vt(1) 260
172.31.204.64/27, apm - Advanced Power Management apm(3) 583
mp3dec - decode audio MPEG files (layers 1, 2 and 3) .......................... apm(8) 830
0a, mp3dec - decode audio MPEG files (layers 1, ....... pushssl - attach SSL version 2 encryption to a communication channel ..................................................... pushssl(2) 493
vt - emulate a VT-100 or 220 terminal ................................................................. vt(1) 260
0a, 1a, 2a, 5a, 6a, 7a, 8a, qa, va - assemblers ... 2(1) 4
0c, qa, va - assemblers .......................... 2(1) 5
01, 1c, 2c, 5c, 6c, 7c, 8c, kc, qc, vc - C compilers ... 2(1) 5
01, 11, 21, 51, 61, 71, 81, k1, ql, vl - loaders ... 2(1) 8
sokoban, sudoku - time wasters ............. 2(1) 8
0a, 1a, 2a, 5a, 6a, 7a, 8a, qa, va - assemblers ... 2(1) 4
0c, 1c, 2c, 5c, 6c, 7c, 8c, kc, qc, vc - C compilers ... 2(1) 5
6to4 tunnel of IPv6 through IPv4 6in4 - configure and run automatic or manual .......................... mk9660(8) 885
01, 11, 21, 51, 61, 71, 81, k1, ql, vl - loaders ... 2(1) 8
0a, 1a, 2a, 5a, 6a, 7a, 8a, qa, va - assemblers ... 2(1) 4
0c, 1c, 2c, 5c, 6c, 7c, 8c, kc, qc, vc - C compilers ... 2(1) 5
01, 11, 21, 51, 61, 71, 81, k1, ql, vl - loaders ... 2(1) 8
dump9660, mk9660 - create an 9660 CD image ................................................................. mk9660(8) 885
systems .................. doxsr, 9660src, a:, b:, c:, d:, 9fat, dosmnt, eject - DOS and ISO9660 file .......................... doxsr(4) 667
srv, srvold9p, 9loads, 9loadsrc, 9loaddsk, ld - PC bootstrap program .......................... srv(4) 713
intro - introduction to the Plan 9 File Protocol, 9loads, 9loadsrc, 9loaddsk, ld - PC bootstrap program .......................... 9load(8) 823
fversion - initialize 9P connection and negotiate version ................................................. fversion(2) 403
allocreq, closereq, lookupreq, removereq - 9P fid, request tracking ................../freereqpool, ..................................................... 9pfid(2) 279
threadlist, threadpostmount, srv 9P file service ................................./respod, responderror, .......................... 9p(2) 273
srv - serve 9P from Unix ................................................................. u9fs(4) 718
mnt - attach to 9P servers ................................................................. mnt(3) 619
9P to text translator 9Pcon(8) 826
bootstrap program ................... 9loads, 9loadsrc, 9loaddsk, ld - PC .......................... aqan - always available network .......................... aqan(8) 827
Web ..................... abaco, readweb - browse the World-Wide .......................... abaco(1) 10
Permuter Index

sin, cos, tan, asin, acos, atan, 
aoserv - serve data via 
aoe = 
interface .................................................. sdao 
process cleanup ...................... exits, exits, atexit, 
notification ................... notify, noted, 
strtol, strtoul, strtoull - convert text to/ atof, 
oiunit - return size of 
cas, casv, casp, loadlink, storecond, _tas- 
connection ............................................. pipefile 
communication channel ............ pushssl 
okThumbprint, readcert, readcertchain 
mnt - 
put2/, /setmap, findseg, unusemap, loadmap, 
stat, wstat - inquire or change file 
controller 
serial, usbeject, usbfat: - Universal Serial Bus/ 
mp3enc - create mp3 
audio - digital 
mp3dec - decode 
/login, noworld, auth_proxy, fault_proxy, 
/amount_getkey, auth_freeAl, auth_chuid, 
convT2M, convm2T2, convT2R2, convm2T2, 
/factotum, fgui - 
- Digital Pathways SecureNet Key remote 
keys, warning - 
login, news, none, as - maintain or query 
server .. - fault - set up 
authsvr, p9any, p9sk1, p9sk2 - 
_asrdresp - routines for communicating with 
/auth_rpc, auth_getkey, amount_getkey, 
auth_chuid, auth_challenge, auth_response, 
/auth_proxy, fault_proxy, auth_allocrpc, 
/auth_userpassword, auth_getuserpassword, 
/auth_allocrpc, auth_freeRPC, auth_rpc, 
for/ 
/auth_response, auth_userpassword, 
amount, news, addns, login, noworld, 
/auth_challenge, auth_response, auth_freshchal, 
/auth_proxy, auth_allocrpc, auth_freeRPC, 
news, 
/printnetkey, status, enable, disable, 
authentication protocols 
/auth_response, auth_freechral, auth_response, 
through IPv4 ............................... 6in4 - configure and run 
systems ....................................... mntgen 
mount, to a port 
them. ............................................. kmap - show a list of 
aan - always 
/insertav, lookupav, deleteav, avwalk, 
exit 
acme, win, 
processing language ................ UP 
ISO9660 file systems ........................ dossrv, 9660srv, a, 
structural regular expressions .......... sam, 
cdfs, cddb - optical disc (CD, DVD, 
backup, backuser, dumparenas, restore - 
and server 
rtcp, stcp - measure TCP 
statusbar - display a 
vacfs - a 
Spam, Reply, Send, Post, Delmesg, Save - file 
/Bprint, Byprint, Bwrite, Bflush, Bterm, 
beautifier, mctrc 
cb - C program 
uptime - show how long the system has 
trace - show (real-time) process 
symoff, fpformat, belee80fts, belee8eats, 
government ............................... ipok - verify that an IP address 
/ge4, ge8, put1, put2, put4, put8, beswab, 
mpassign, mprand, sttrmp, mpmft, mpto, 
drawrepl, replclipr, line, poly, fillpoly, bezier, 
atan2 - trigonometric functions 
ATA-over-EtherNet (A0) ........................... aoe 
ATA-over-EtherNet (A0) interface .............. aoe 
ATA-over-EtherNet (A0) storage device 
axexitd, terminate - terminate process, 
ationfify - handle asynchronous process 
atoi, atol, atol, charstod, strtod, 
atomic I/O unit for file descriptor 
atomic RMW operations ...................... ainc, adec, 
attatch, auth, messages to establish 
attach filter to file in name space 
pipefile 
attach SSL version 2 encryption to a 
attach TLS1 or SSL3 encryption to a/ 
apptachproc, get1, get2, get4, get8, put1, 
attribution ...................... stat 
audio - digital audio jukbox 
audio - SoundBlaster or ESS1688 audio 
audio, ccid, disk, ether, k8, print, probe, 
audio files 
mp3enc(1) 
audio jukbox 
mp3dec(1) 
avoid MEGP files (layers 1, 2 and 3) 
auth - messages to establish a connection 
auth_allocrpc, auth_freeRPC, auth_rpc, 
/auth_challenge, auth_response, 
authdial, passstkey, nvcsum, readnram, 
authentication agent 
/authorization box - securenet 
authentication database files ...... /debug, wrkey, 
authentication on a file descriptor to a file 
authentication protocols 
authsvr(6) 
authentication servers ................... /asgettck 
auth_freeal, auth_chuid, auth_challenge, 
auth_freechal, auth_response, 
auth_freeRPC, auth_rpc, 
/auth_getinfo - routines for authenticating/ 
auth_getkey, amount_getkey, auth_freeal, 
auth_getuserpassword, auth_getinfo - routines 
/auth_proxy, fault_proxy, auth_allocrpc, 
auth_response, auth_userpassword, 
automatic or manual 6to4 tunnel of IPv6 
automatically generate mount points for file 
mntgen - fileserver: - mountpoint - configure 
available keyboard maps and switch between 
available network 
avnext, avperv, endwalk - AVL tree routines 
await, wait, waitpid - for a wait to process for 
awd - interactive text windows 
awk - pattern-directed scanning and 
daasrv(4) 
be, bc, d, de, dosmnt, eject - DOS and 
B, sam.save, samterm - screen editor with 
B) track reader and writer file system 
backup venti arenas to blu-ray discs or/ 
band2 = Viaduct bridge-and-tunnel client 
bandwidth 
bar graph status window 
 baised file system 
based mail reader ....... /mailpullb, msgb, Arch, 
basename - strip file name affixes 
BBuffered - buffered input/output 
be - arbitrary-precision arithmetic language 
beautifier, mctrc 
been running 
beebefdos, leiene80fts, leieneeats, 
belongs to a country approved of by the US 
bewal, bewas, leswal, leswal, leswaw 
vbetomp, mbetomp, letompt, mbtople, mbtou, 
bezspine, fillbeizer, fillbezspine, ellipse, 
beq
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>bFCBDecrpyt</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>/Bungetc, Bungetre, Bread, Bseek, Boffset, Bputc, Bputre, Bprint, Bwrive, Bwrite, Bwrite, Bwrite</td>
<td>Encrypts files with a biometric key.</td>
<td>(309)</td>
</tr>
<tr>
<td>/Bopen, Binit, Binit, Bdrline, Brdstr, allocation</td>
<td>Encrypts files with a biometric key.</td>
<td>(309)</td>
</tr>
<tr>
<td>strip</td>
<td>remove symbols from files</td>
<td>(224)</td>
</tr>
<tr>
<td>apm</td>
<td>Advanced Power Management 1.2</td>
<td>(583)</td>
</tr>
<tr>
<td>apm</td>
<td>Advanced Power Management 1.2</td>
<td>(830)</td>
</tr>
<tr>
<td>light, pencil, keyboard, params, prompter, /Bungetre, Bread, Bseek, Boffset, Bfilides, vtglovtablocalo, vtalgottglobalo, Venter, VFigeroextend, Vtvecroscore, Vttnup, setupDEStaste, des_key_setup, /infatizblblock, flateerr, mkcrctab, sum, md5sum, shalsum</td>
<td>sum and count batch files</td>
<td>(2013)</td>
</tr>
<tr>
<td>bFCBDecrpyt</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>/dumparenas, restore – backup venu arenas to toico – view and convert</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>/Bputc, Bputre, Bprint, Bwrite, Bwrite</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>/Bopen, Binit, Binits, Bdrline, Brdstr</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>tel</td>
<td>iwhois – look in phone</td>
<td>(312)</td>
</tr>
<tr>
<td>cpurc, cpurc.local, termrc, termrc.local – bootwinnt, personalize, setup.9fat</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>dhcpcd, dhcpleases, rarpd, ttfdp</td>
<td>Internet control</td>
<td>(312)</td>
</tr>
<tr>
<td>init</td>
<td>initialize machine upon</td>
<td>(312)</td>
</tr>
<tr>
<td>personalize, setup.9fat,</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>bootloopy</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>9load, 9xlabel, 9loadusb, 9loadask, ld – PC</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>booting</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>bootloopy, bootplan9, bootwin9x</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>Bgetre, Bgeted, Bungetc, Bungetre, /ellipse, filllellere, arc, fillarc, icoss, icossin2, /publicfscreen, freescreen, alloowindow, Pathways SecureNet Key remote authentication</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>Boffset, Bfilides, Blineden, Bputc, Bputre, /Bread, Bseek, Boffset, Bfilides, Blineden, Bgetre, Bputc, Bputre, /Bopen, Binit, Binits, /Bgetc, Bgeted, Bgetc, Bgetre, Bgeted, Bdrlineset, getfelds, tokenize, getnetconninfo, freenetconninfo</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>bandt2 – Viaduct</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>findviaduct</td>
<td>look up data about about Viaduct</td>
<td>(312)</td>
</tr>
<tr>
<td>kill, slay</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>abaco, readweb – dict – dictionary</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>/Bgetre, Bgeted, Bungetc, Bungetre, Bread, Bputc, Bputre, Bprint, Bwrite, Bflush</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>Computing (VN)</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>Bprint, Bwrite, Bflush, Bterm, Bbuffersered</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>fseek, rewind, feof, ferror, clearrer – standard packetrouter, packtrr – zero-ency network</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>/getwindow, gengetwindow, flushimage, /lockdisplay, unlockdisplay, op_Font, fnt, fntreasn, fntbloom, fntindex, fntset</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>/Brdstr, Bgetc, Bgetre, Bgeted, Bungetc, compress and expand</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>gzip, gunzip, bzip2, ppci</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>usbd – Universal Serial</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>/Blineden, Bputc, Bputre, Bprint, Bwrite, enc32, dec16, enc16, encodefm - decoding</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>/Memparkparam, memimagineit, wordaddr, /unloadimage, readimage, writetimeimage, unzip – compress and expand</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
<tr>
<td>asiteze – translate Plan 9 C to ANSI</td>
<td>Encrypts files with a biometric key.</td>
<td>(312)</td>
</tr>
</tbody>
</table>
Permutation Index

style \(\rightarrow\) Plan 9 coding conventions for
consoles, pcc \(\rightarrow\) APE
0c, 1c, 2c, 5c, 6c, 7c, 8c, kc, qc \(\rightarrow\) file systems
0d, 9fat, dosmt, eject \(\rightarrow\) DOS and ISO9660
C compilers
C compiler driver
C language preprocessor
C program beautifier
cpp \(\rightarrow\) translate Plan 9
tvglobalatolocal, tvmallocoglob \(\rightarrow\) Venti block
Fontchar, Font \(\rightarrow\) font utilities
caches

cws \(\rightarrow\) flush instruction and data

dc \(\rightarrow\) desk

maloc, mallocalign, mallocz, free, realloc
rtrace \(\rightarrow\) trace process system
tcp17009, tcp17010, tcp17013 \(\rightarrow\) listen for
tracepipe \(\rightarrow\) forward incoming
to data/ Intmap, allocmap, freemap, insertkey,
lock, canlock, runlock, wlock, / lock,
multid, divt, rectadpt, rectsubpt, inserted,
lock, / lock, canlock, unlock, qlock,
quaccess, qlock, qunlock, qunlock,
canerror, qlock, pinlock, rectlock, rectlock,
quaccess, qlock, canlock, runlock,
quaccess, wunlock, rsleep, rwaclck,
cap \(\rightarrow\) capabilities for setting the user id of
card
scat \(\rightarrow\) sky

usbeject, usbfail \(\rightarrow\) Universal Serial Bus / audio,
system \(\rightarrow\) cdfs, cdbb \(\rightarrow\) optical disc
whatis, \(\rightarrow\) command language \(\rightarrow\) rc,
dump9660, mk9660 \(\rightarrow\) create an ISO-9660
reader and writer file system

functions \(\rightarrow\) fabs, fmod, floor,
alt, chanclose, chancreate, chanfree, chaninit,
stat, wstat \(\rightarrow\) inquire or
chgrp \(\rightarrow\) seek
brk, sbrk
sebgbrk
chmod
bind, mount, umount
bind, mount, umount
password, netkey
chdir

replica management

printnetkey, status, enable, disable, authsvr,
alt, chanclose, chancreate, chanfree, pipe
create an interprocess

SSL version 2 encryption to a communication
TLS1 or SSL3 encryption to a communication
/chancreate, chanfree, chaninit, chanclose,
/buildfont, freefont, Pfmt, Rfmt, strochanc,
totellure, toupper \(\rightarrow\) ASCII
fprint \(\rightarrow\) print histogram of
scribblealloc, recognize
UTF, Unicode, Ascii, rune

character sets
doquote, needsqlquote \(\rightarrow\) quoted
ascii, unicode \(\rightarrow\) interpret ASCII, Unicode
keyboard \(\rightarrow\) how to type
tr \(\rightarrow\) translate
- convert text to/

- atof, atoi, atol, atof, utfcpcy, utflen, utfnlen, utfplane, \(\rightarrow\) runetochar,

- style(6) \(\rightarrow\) 794
- consolefs(4) \(\rightarrow\) 661
- pcc(1) \(\rightarrow\) 162
- C compilers
- C compiler driver
- C language preprocessor
- C program beautifier
- cpp(1) \(\rightarrow\) 51
- dosrsv, 9660rsv(4) \(\rightarrow\) 667
- dosrsv(1) \(\rightarrow\) 40
- Globatolocal, tvmallocoglob(2) \(\rightarrow\) Venti cache
- vtechwrote, write, vfactory(2) \(\rightarrow\) 559
- cache(4) \(\rightarrow\) 658
- cachechars, agefont, loadchar, Subfont,
cashed-worm file server, dump
- cachefiles(4) \(\rightarrow\) 665
- segflush(2) \(\rightarrow\) 527
- calc \(\rightarrow\) print calendar
- dc(1) \(\rightarrow\) 62
- calendar \(\rightarrow\) print upcoming events
- calend(1) \(\rightarrow\) 38
- malloc(2) \(\rightarrow\) 448
- rattrace(1) \(\rightarrow\) 180
- map \(\rightarrow\) read - catenate files
- cat(1) \(\rightarrow\) 39
- catalogue and Digitized Sky Survey
- scat(7) \(\rightarrow\) 818
- bc \(\rightarrow\) C program beautifier
- cb(1) \(\rightarrow\) 40
- cd, cd, ether, kb, print, probe, serial,
- cdfs(4) \(\rightarrow\) 656
- mk9660(8) \(\rightarrow\) 885
- cdc, cdbb \(\rightarrow\) optical disc (CD, DVD, B) track
- mk9660(8) \(\rightarrow\) 842
- cec \(\rightarrow\) Coerait Ethernet Console
- ece(2) \(\rightarrow\) 384
- ceil \(\rightarrow\) absolute value, remainder, floor, ceiling
- ceil(2) \(\rightarrow\) 384
- char \(\rightarrow\) cdfs file system
- cdfs(4) \(\rightarrow\) 656
- chanclosing, chancprint, mainstactsize,
- thread(2) \(\rightarrow\) 545
- change file attributes
- stat(5) \(\rightarrow\) 747
- changefile group
- chgrp(1) \(\rightarrow\) 41
- change file offset
- seek(2) \(\rightarrow\) 523
- change memory allocation
- brk(2) \(\rightarrow\) 313
- segbrk(2) \(\rightarrow\) 526
- chmod(1) \(\rightarrow\) 42
- chancopy
- bind(1) \(\rightarrow\) 43
- changename
- bind(1) \(\rightarrow\) 33
- changenew
- bind(2) \(\rightarrow\) 307
- change or verify user password
- passwd(1) \(\rightarrow\) 159
- changetraining directory
- chdir(2) \(\rightarrow\) 317
- change upset, push, scan - client-server
- auth(8) \(\rightarrow\) 833
- changer"es, conkeys, conkeys2,
- thread(2) \(\rightarrow\) 545
- chaninit, chanclosing, chancprint,
- thread(2) \(\rightarrow\) 545
- channel
- pipe(2) \(\rightarrow\) 479
- channel
- pushssl-t attach
- pushssl(2) \(\rightarrow\) 493
- channel
- readerr, readerrchKeep
- pushssl(2) \(\rightarrow\) 494
- character class
- thread(2) \(\rightarrow\) 445
- character class
- ttypes(2) \(\rightarrow\) 339
- character frequencies
- freq(1) \(\rightarrow\) 91
- character recognition
- scribble(2) \(\rightarrow\) 517
- character set and format
- utf8(6) \(\rightarrow\) 797
- character set
- tcss(2) \(\rightarrow\) 233
- character strings
- quote(2) \(\rightarrow\) 501
- characters
- ascii(1) \(\rightarrow\) 25
- characters
- keyboard(6) \(\rightarrow\) 768
- characters
- tr(1) \(\rightarrow\) 242
- charset, strc, strl, strll, strtl, strth, strtlw
- ato(2) \(\rightarrow\) 297
- charsequence, runelen, runelen, fullrun,
- rune(2) \(\rightarrow\) 514
- chdir \(\rightarrow\) change working directory
- chdir(2) \(\rightarrow\) 317
assert -
fmtindex, fmtree, buildindex, checkarenas,
vtmalo2c, vtlraocc, vtsrudp, vfree -
times, cycles - cpu time in this process and
client -
aquarela -
serial interface (TWS) and inter-integrated
pump - copy asynchronously via a large
localaddr, symoff, fpformat, beee280ftos,
strlen, ...; strcat, strncat, strpncpy,
strtok, strup, struprc, struprc, strupstr,
/opendevdata, openep, sパーテ, unstuff,
totterlune, toupperre - Unicode character
toller, toupper, tolower - ASCII character
finddevs, loaddevstr, matchdevcs, /usbcm,
terminate - terminate process, process
/fetk, fsetpos, fseek, rewind, seek, ferror,
cifs - Microsoft™ Windows network filesystem
nsf - Sun network file system
tlsrvtunnel, tlsclienttunnel - TLS server
and tunnel
vtping, vtrpc, ventidoublechecksh1 - Venti
band2 - Viatduct bridge-and-tunnel
up data about Viatduct bridging-and-tunneling
read, write, copy - simple Venti
changes, pull, push, scan -
/applog, compactdb, updatedb - simple
/namedimage, setalpa, loadimage,
freememimage, memsetchun, loadmemimage,
date,
rtc - real-time
/cron -
timesync - synchronize the system
reboot, etc. ...; cons - console,
consolas, C,
create file -
Control, Controlset, activate, closecontrol,
loaddevstr, ...; usbcmd, classname,
/removefile, walfkle, opendirfile, readdirfile,
initdir, genwindow, drawrow, initdisplay,
Fid, Fidpool, alcofpool, freefipool, alcofpool,
Tree, alcoftree, freetree, File, createfile,
ioopen, ioproc, ioread, ioreadh, iowrite - /
...; menuhist, setcursor - /
...; initmouse, readmouse,
add3, sub3, neg3, div3, mul3, eqpt3, /Reapprool, allocreqpol, freereqpol, allocreqpol,
SCSI device operations -
open/sci,
maps ...; cma2rgb, lookupcmd - control message parsing ...
src - find source
/style - Plan 9
/bind -
getmap, colors - display
RGB, readcolormap, writecolormap - access display
cmap2rgb, cmap2rgb, rgb2cmapi - colors and colors - representation of pixels and colors
cmap2rgb, cmap2rgb, rgb2cmapi - colors and colors - display color map
col -
eqrect, ptnrect, rectxrect, rectsclp,
sorted files -
mailcmd - mail the output of a failed
time - time a
exec, exit, flag, fork, shift, wait, whatis, ...; -
doctype - intu
check program invariants ...
checkindex, conf, fma2renas, fmbloom, ...; checking
memory allocators, vbrk, vtmalo2c, chgrp - change file group
children - cputime, chmod - change mode
cifs - Microsoft™ Windows network filesystem
CIFS server
client ...
circuit (PC) interface - twsi - two-wire
factory buffer -
cistrace,iscrwsc,iscwfrz,iscfrw
client, cistrcmap, cistrcpcp, cistrctnp, cistrctnp
str: cistrctnp, cistrctnp, cistrctnp, cistrctnp, cistrctnp
linkstr - string operations ...
/vstrh, /strcrh, /strcrh, /strcrh, /strcrh,
client and server ...
clint ...
client-server replica management ...
client-server replica management ...
loadimage, unloadimage, readimage, /
loadmemimage, unloadmemimage, /
clock - date and time
/cron -
timesync - a time source
clowns, process/process group ids, user, null,
clock - file system for console access
/close - open a file for reading or writing,
/closecontrolset, controlalled, controlwire, /
closedev, configdev, devctl, finddevs,
closedir, hasperm - in-memory file/
closedisplay, getdefont, getwindow, /
lockfile, lookupid, removelf, Req, Reqpol, /
closedir, movefile, walfkle, openfile, /
closeioproc, iocall, iocall, iointerrupt, iodal,
/keyboard, /kboardcontrol /
closemouse, moveto, getrect, drawgetrect, /
closeopt, dot3, cross3, len3, dist3, unit3, /
closereq, lookupere, removeq - 9f fid, /
closecli, csiready, csci, csci, csci, csci, csci, csci, csci - stick a fid
/cmap2rgb, cmap2rgb, colors and colors
Cmdbuf, parsecmd, respondcmderror,
cmp - compare two files
code for executable ...
coding conventions for C
col - column alignment
collect files for distribution ...
color - representation of pixels and colors
color map ...
color map ...
color maps ...
colors ...
color maps ...
colors ...
colors ...
colors ...
colors ...
colors ...
colors ...
colors ...
colors ...
combinerect, Dx, Dy, Pt, Rect, Rpt - /
eqct, command - select or reject lines common to two
command ...
command ...
command language ...
rc, cd, eval,
command line for formatting a document ...
doctype(1)
Permitted Index

```plaintext
exponential, logarithm, power, square root .
uncompress, zip, unz - compress and
aliasmail - at, drain,
frexp, ldexp, modf - split into mantissa and
exp, log, log10, pow, pow10, sqrt -
exponent
exp, log10, pow, pow10, sqrt -
exp, log, log10, pow, pow10, sqrt -
regsub, regexec, regsub, regerror - regular
regexp - regular
samterm - screen editor with structural regular
fs,
crtpre, crtin, crtouet, crtdfree, cttrefree -
s_grow, s_read, s_read_line, s_getline -
font, subfont -
image -
/vxls2txt, olesfs, mswordstrings, msexceltables -
remainder, floor, ceiling functions -
tweak - edit image files, subfont files,
mug - convert an image to a
large primes -
delkey - delete keys from
mailcmd - mail the output of a
cap -
/setbuf, fgetpos, ftell, fsset, fseek, rewind,
sokaban, sudoku - time wasters .
s, Ss, ap -
call -
points/ -/pdlstix, vdlv3, vrem3, pn2F3, ppp2F3,
/setbuf -
/fget, fgetchar, fputc, putc, putchar, ungetc,
factotum,
clunk - forget about a
open, create - prepare a
closefd, lookupfd, removefd, Req, / -
Fid, gettokens, tokenize - break a string into
access - determine accessibility of
dd - convert and copy a
exec, _privates, _nprivates, _tos - execute a
fortune - sample lines from a
httpfile - establish a web
namespace - name space description
open - open a file for reading or writing, create
prepare a fid for I/O on an existing or new
pr - print
readn, write, pread, pwrite - read or write
read, write - transfer data from and to
remove, remove - remove a
shalsum - sum and count blocks in a
swap - establish a swap
tail - deliver the last part of a
unix - report repeated lines in a
venti.conf - a venti configuration

exp, log, log10, pow, pow10, sqrt -
exponent
exp, log10, pow, pow10, sqrt -
exponent
exp, log10, pow, pow10, sqrt -
exponent
expression . . . /regcompl, regcompl, regexec,
expression notation . . .
expr - file server maintenance
ext2srv - ext2 file system
extended precision arithmetic . . . /mpmagsub,
tensible strings . . . . . . /s_putc, s_unique,
external format for fonts and subfonts .
external format for images .
external keyboard input
extract a vac archive on Venti
extract printable strings
extract printable text from Microsoft/
fabs, fmod, floor, cell - absolute value
face - face files
face files, etc -
face icon -
facades, seemail, whois - mailbox interface
factor, primes - factor a number, generate
factotum -
factotum, fgui - authentication agent
mailcmd() -
ecp -
/setbuf, fgetpos, ftell, fsset, fseek, rewind,
sokaban, sudoku - time wasters .
s, Ss, ap -
call -
points/ -/pdlstix, vdlv3, vrem3, pn2F3, ppp2F3,
/setbuf -
/fget, fgetchar, fputc, putc, putchar, ungetc,
factotum,
clunk - forget about a
open, create - prepare a
closefd, lookupfd, removefd, Req, / -
Fid, gettokens, tokenize - break a string into
access - determine accessibility of
dd - convert and copy a
exec, _privates, _nprivates, _tos - execute a
fortune - sample lines from a
httpfile - establish a web
namespace - name space description
open - open a file for reading or writing, create
prepare a fid for I/O on an existing or new
pr - print
readn, write, pread, pwrite - read or write
read, write - transfer data from and to
remove, remove - remove a
shalsum - sum and count blocks in a
swap - establish a swap
tail - deliver the last part of a
unix - report repeated lines in a
venti.conf - a venti configuration
```

14-i
Permuted Index
and html . . . . . . . . . . . . . . . . . . . . ms2html,
/HRange, HSPairs, hmydomain, hversion,
/HContent, HContents, HETag, HFields, Hio,
fmt,
webcookies 
/hreqcleanup, hrevhfields, hrevspairs, hstrdup,
HTTP server . . . . . . . . . . . . . . . . . . . . . . . . . .
/hrevhfields, hrevspairs, hstrdup, http11,
/HContents, HETag, HFields, Hio, Htmlesc,
/hrevspairs, hstrdup, http11, httpfmt,
/ /httpunesc, hunallowed, hungetc, hunload,
/HttpReq, HRange, HSPairs, hmydomain,
/hungetc, hunload, hurlfmt, hurlunesc,
/getmemdefont, memimagestring, iprint,
/hungetc, hunload, hurlfmt, hurlunesc, hvprint,
sinh, cosh, tanh 
Interface Association (PCMCI) device . . . . . . . .
interface (TWS) and inter-integrated circuit
protocols over IP . . . . . . . . ip, esp, gre, icmp,
convert/ . jpg, gif, png, ppm, bmp, v210, yuv,
mug - convert an image to a face
crop,
/ellipse, fillellipse, arc, fillarc, icossin,
cap  capabilities for setting the user
invertmat, xformpoint, xformpointd,/ . . . . . . .
pcmcia 
/leieee80ftos, leieeesftos, leieeedftos,
crop, iconv  frame, crop, and convert
dump9660, mk9660  create an ISO-9660 CD
replclipr, line, poly, fillpoly, bezier,/ . . . . . . . .
tweak  edit
typesetter output files . . . . . page  view FAX,
mug - convert an
httpd, save,
 allocating, freeing, reading, writing
image  external format for
 drawing routines for memory-resident
memltorearn  windows of memory-resident
pop3,
system . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
trampoline  forward
/wunlock, rsleep, rwakeup, rwakeupall,
/beswal, beswav, leswab, leswal, leswav 
/leieeesftos, leieeedftos, ieeesftos, ieeedftos 
functions . . . . . . . . . . . . . . . . . . . . . . . NaN,
/deflatezlibblock, inflateinit, inflate, inflatezlib,
cs, csquery, dns, dnstcp, dnsquery, dnsdebug,
astro  print astronomical
arch  architecture-specific
keys.who  biographic
/ctlprint, deactivate, freectlfont, freectlimage,
Display, Point, Rectangle, Cursor,
fversion 
init 
keyboard control . . . . . . . . . . . . . . . . . . . . . .
getrect, drawgetrect, menuhit, setcursor / . . .
okThumbprint,/ . pushtls, tlsClient, tlsServer,
fscanf, scanf, sscanf, vfscanf  scan formatted
kbin  external keyboard
 preprocess so inclusion commands in troff
fgets, gets, fputs, puts, fread, fwrite  Stdio
Bwrite, Bflush, Bterm, Bbuffered  buffered
feof, ferror, clearerr  standard buffered
stat, wstat 
avlnext, avlprev, endwalk - AVL/ . . mkavltree,
 integer to data/ Intmap, allocmap, freemap,

20-i

html2ms  convert between troffs ms macros .
ms2html(1)
htmlesc, halloc, hbodypush, hbuflen,/ . . . . . .
httpd(2)
Htmlesc, HttpHead, HttpReq, HRange,/ . . . . . .
httpd(2)
htmlfmt  simple text formatters . . . . . . . . . .
fmt(1)
htmlroff  HTML formatting and typesetting . .
htmlroff(1)
htmlroff  HTML formatting and typesetting . .
htmlroff(6)
HTTP cookie manager . . . . . . . . . . . . . . . . . . webcookies(4)
http11, httpfmt, httpunesc, hunallowed,/ . . . .
httpd(2)
httpd, save, imagemap, man2html, webls  . . .
httpd(8)
httpfile  serve a single web file . . . . . . . . . . .
httpfile(4)
httpfmt, httpunesc, hunallowed, hungetc,/ . . .
httpd(2)
HttpHead, HttpReq, HRange, HSPairs,/ . . . . . .
httpd(2)
httpunesc, hunallowed, hungetc, hunload,/ . .
httpd(2)
hurlfmt, hurlunesc, hvprint, hwrite, hxferenc, .
httpd(2)
hversion, htmlesc, halloc, hbodypush,/ . . . . . .
httpd(2)
hvprint, hwrite, hxferenc,  routines for/ . . . .
httpd(2)
hwdraw  drawing routines for/ . . . . . . . . . . .
memdraw(2)
hwrite, hxferenc,  routines for creating an/ . .
httpd(2)
hyperbolic functions . . . . . . . . . . . . . . . . . . .
sinh(2)
hypot  Euclidean distance . . . . . . . . . . . . . . .
hypot(2)
i82365  Personal Computer Memory Card . . .
i82365(3)
IrC) interface . . . . . . . . twsi - two-wire serial
twsi(3)
icmpv6, ipmux, rudp, tcp, udp  network . . . .
ip(3)
ico, togif, toppm, topng, toico  view and . . . .
jpg(1)
icon . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
mug(1)
iconv  frame, crop, and convert image . . . . . .
crop(1)
icossin2, border, string, stringn, runestring,/ .
draw(2)
id of processes . . . . . . . . . . . . . . . . . . . . . . .
cap(3)
ident, matmul, matmulr, determinant, adjoint,
matrix(2)
identify a PCMCIA card . . . . . . . . . . . . . . . . . .
pcmcia(8)
idiff  interactive diff . . . . . . . . . . . . . . . . . . .
idiff(1)
ieeesftos, ieeedftos  machine-independent/ .
debugger(2)
ike  IPsec Internet Key Exchange file server . .
ike(4)
image . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
crop(1)
image . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
mk9660(8)
image  external format for images . . . . . . . .
image(6)
Image, draw, gendraw, drawreplxy, drawrepl, .
draw(2)
image files, subfont files, face files, etc. . . . . .
tweak(1)
image, graphic, PostScript, PDF, and . . . . . . . .
page(1)
image to a face icon . . . . . . . . . . . . . . . . . . . .
mug(1)
imagemap, man2html, webls  HTTP server . . .
httpd(8)
images . . . . . . . . /bytesperline, wordsperline allocimage(2)
images . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
image(6)
images . . . /memimagestring, iprint, hwdraw
memdraw(2)
images . . . . . . . /memltofrontn, memltorear,
memlayer(2)
imap4d  Internet mail servers . . . . . . . . . . . .
pop3(8)
import  import a name space from a remote .
import(4)
incoming calls to another address . . . . . . . . . trampoline(8)
incref, decref  spin locks, queueing/ . . . . . . .
lock(2)
independent access to executable files . . . . . .
mach(2)
independent debugger functions . . . . . . . . . .
debugger(2)
Inf, isNaN, isInf  not-a-number and infinity . .
nan(2)
inflateblock, inflatezlibblock, flateerr,/ . . . . . .
flate(2)
inform  network database . /mkdb, mkhosts,
ndb(8)
information . . . . . . . . . . . . . . . . . . . . . . . . . .
astro(7)
information and control . . . . . . . . . . . . . . . . .
arch(3)
information for key holders . . . . . . . . . . . . . .
keys.who(6)
init  initialize machine upon booting . . . . . . .
init(8)
initcontrols, namectlfont, namectlimage,/ . . . .
control(2)
initdraw, geninitdraw, drawerror, initdisplay,/ .
graphics(2)
initialize 9P connection and negotiate version .
fversion(2)
initialize machine upon booting . . . . . . . . . . .
init(8)
initkeyboard, ctlkeyboard, closekeyboard  . . .
keyboard(2)
initmouse, readmouse, closemouse, moveto, .
mouse(2)
initThumbprints, freeThumbprints, . . . . . . . . .
pushtls(2)
input . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
fscanf(2)
input . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
kbin(3)
input . . . . . . . . . . . . . . . . . . . . . . . . . soelim
soelim(1)
input and output . . . . /putc, putchar, ungetc,
fgetc(2)
input/output . . . . /Bputrune, Bprint, Bvprint,
bio(2)
input/output package . . . . . . /fseek, rewind,
fopen(2)
inquire or change file attributes . . . . . . . . . . .
stat(5)
insertavl, lookupavl, deleteavl, avlwalk, . . . . . .
avl(2)
insertkey, caninsertkey, lookupkey, deletekey .
intmap(2)

141
427
427
89
108
763
728
427
871
686
427
427
427
427
427
427
452
427
531
432
605
643
606
111
148
54
350
589
450
904
109
340
687
54
885
766
350
247
157
148
871
289
766
452
456
917
688
953
442
445
340
468
382
893
806
584
770
873
321
411
403
873
441
461
494
400
614
211
380
309
388
747
304
433


msexceltables – extract printable text from.
cifs -
closept3, dot3, cross3, len3, dist3, unit3, tagrd – plumb a
files
.
dump6600,
avlwalk, avlnext, avlperv – AVL tree/
/inflateblock, inflatezlibblock, deflate,
dnsquery, /query, ipquery, mhash,

mkfs, flash memory

/writesfont, stringsfont, strsubfontwidth,
archfs – mount
dnsdebug, /query, ipquery, mhash, mckd, system

points for file systems

chmod – change
spin – verification tool for
frexp, ldexp, touch – set
mtime – print file
vgadb – VCA controller and
winwatch –
cpiofs, tapfs, tarfs, tfps, vfs, v10fs, zipfs –
archfs – stub – provide
mptgen – automatically generate
snap, snaps, create and
bind, mptgen, mount – change name space
bind, mptgen, mount – change name space
winwatch –
mount archival file systems – 32vfs,

mount mpts–style archive

mount point stubs

mount points for file systems

mount process snapshots

mount, umount – change name space

bind, mptgen, mount – change name space

mouse, aux/accpoint – configure a mouse to
mouse control – /closemouse, moveto,
mouse, cursor – kernel mouse interface

Mouse, Menu – graphics events

mouse to a port

move files

move, xform, xform, persp, look, viewport – /
moveto, getread, getdraw, getmenu/

mount, umount – change name space

mv3dec – decode audio

MPEG files (layers 1, 2 and 3)

MPEG files (containers, 29)

MPEG files (layers 1, 2 and 3)

Microsoft documents . /olefs, mswordstrings,

Microsoft™ Windows network filesystem client

midptr, lerp3, reflect3, nearseg3, pldist3, /

Mifare Ultralight tag

mk, membername – maintain (make) related

mk9660 – create an ISO-9660 CD image

mk9660(8) –

mkavtire, insertavl, lookupavl, deleateavl,

mkcrctab, blockccrd, adler32 – deflate/

mkdb, mkhosts, cs, csquery, dns, dnsctp,

mkdir – make a directory

mkext – archive or update a file system

mkfastfs – make a journalling file system for

mkfont – subfont manipulation

mkfs – make a file system

mkfs–style archive

mkhosts, cs, csquery, dns, dnsctp, dnsquery,

mkpipe – make a compressed read–only file

mksafs – make a compressed file system

mktemp – make a unique file name

ml, mlimgr, mliw – unmoderated mailing lists

mhingo – macros for typesetting Japanese

mnt – attach to 9P servers

mntgen – automatically generate mount

mptgen, mount – change name space

mouse, aux/accpoint – configure a mouse to
mouse control – /closemouse, moveto,
mouse, cursor – kernel mouse interface

Mouse, Menu – graphics events

mouse to a port

move files

move, xform, xform, persp, look, viewport – /
moveto, getread, getdraw, getmenu/

mount, umount – change name space

mv3dec – decode audio

MPEG files (layers 1, 2 and 3)

Microsoft documents . /olefs, mswordstrings,
Permutated Index
telco, faxreceive, faxe send, fax, telcofax, execution, and XMODEM file transfer ... con, remote access daemons

screenlock - disable access to a vt - emulate a VT-100 or VT-220

exit, exit, exeit, exitident, terminate -
cpurc, cpurc.local, termrc, ipoks -
syscall -
screanmail, 

iced, ip - troff and
frdawse, frdawse0, frgetmouse - frames of ed -
fmt, htmlfmt - simple troff, nroff, dpos -
/msexctables - extract printable
strtod, strtol, strtoll, strtof, convert

9pcon - 9 P

acme, win, awd - interactive acme - control files for

syminit, getsym, symbase, pc2sp, pc2line, /getauto, findsym, localsym, globalsym, dhcpd, dhcpdclasses, rarpd, approved of by the US ... ipok - verify

venti arenas to blu-ray discs or restore from available keyboard maps and switch between

timerrecall, recvt, sendt - worker
/nbrecurso, nsend, nssend, nsdest, 
fnterror, silenterror - exception handling for

ioread, iowrite - slave I/O processes for
/nssndp, nsdend, nsdisnet, threaddata, 
/threadcreate, threaddata, threadexit, 
/threadkillgrid, threadmain, threadnotify, 
/threadexitall, threadgetname, threadname, 
/theadinit, threadmain, threadkill, 
/theadt, threadnotify, 
syscall, 

run automatic or manual 64k tunnel of IPv6

astime, tm2sec, timezone - convert date and
date, clock - date and
time - print file modification

gpsfs, gpsservermore - GPS

rtc -
cputime, times, cycles - cpu

nanoseconds since epoch -

trace - show
timesync - synchronize the system clock to a

life, mahjong, memo, sokon, sudo -
management ... workerdispatch, timedispatch, children,
time source -

time, localtime, gmttime, asctime, tm2sec,

tlsrv, tlcclient, tlsrvtunnel, 
okThumpnt, readcert, ... pushits, tlcClient, 
ctime, localtime, gmttime, asctime, tm2sec,

tcp11, tcp12, tcp13, tcp14, tcp15, tcp564, 

tcp17007, tcp17008, tcp17009, tcp17010, tcp565, tcp566, tcp57, tcp993, tcp995, tcp110, tcp113, tcp23, tcp25, tcp53, tcp113, tcp143, tcp15, tcp564, tcp565, tcp566, tcp53, tcp110, tcp113, 

listen, listen1, tcp7, tcp9, tcp23, tcp25, tcp53, tcp113, tcp143, tcp15, tcp564, tcp565, tcp566, tcp53, tcp110, tcp113, 

listen, listen1, tcp7, tcp9, tcp23, tcp25, tcp53, tcp113, tcp143, tcp15, tcp564, tcp565, tcp566, tcp53, tcp110, tcp113, 

tcp11, tcp12, tcp13, tcp14, tcp15, tcp564, tcp565, tcp566, tcp110, tcp113, 

tcp11, tcp12, tcp13, tcp14, tcp15, tcp564, tcp565, tcp566, tcp110, tcp113, 

listen, listen1, tcp7, tcp9, tcp23, tcp25, tcp53, tcp113, tcp143, tcp15, tcp564, tcp565, tcp566, tcp53, tcp110, tcp113,/tcp11, tcp12, tcp13, tcp14, tcp15, tcp564, tcp565, tcp566, tcp110, tcp113, 

tcp11, tcp12, tcp13, tcp14, tcp15, tcp564, tcp565, tcp566, tcp110, tcp113, 

listen, listen1, tcp7, tcp9, tcp23, tcp25, tcp53, tcp113, tcp143, tcp15, tcp564, tcp565, tcp566, tcp53, tcp110, tcp113, tcp110, tcp113, 

ctime, localtime, gmttime, asctime, tm2sec,

tlsrv, tlcclient, tlsrvtunnel, 
okThumpnt, readcert, ... pushits, tlcClient, 
ctime, localtime, gmttime, asctime, tm2sec,
lock – run a command
fgetc, getc, getchar, fpsect, pcsect, putchar,
asci, acscl, utf8
form – make a
/tolowererr, tolowererr, lowererr, touppererr –
asci, uncode – interpret ASCII,
freesubfont, installsubfont, lookupsubfont,
mktemp – a
mktemp, mktemp, mktemp, mktemp, mktemp,
kb, print, probe, serial, usbsubject, usbfat –
usfs – serve 9P from
ssh – secure login and file copy from/to
/sysext, sysext, sysext, sysext, sysext,
/export3, cgroup, dot3, dot3, dot3, dot3,
usbd
/pngimage, pngimage, pngimage, pngimage,
ifconfig, ifconfig, ifconfig, ifconfig,
usbd
/pngimage, pngimage, pngimage, pngimage,
ifconfig, ifconfig, ifconfig, ifconfig,
usbd
/pngimage, pngimage, pngimage, pngimage,
ifconfig, ifconfig, ifconfig, ifconfig,
usbd
/pngimage, pngimage, pngimage, pngimage,
ifconfig, ifconfig, ifconfig, ifconfig,
This book was typeset by the authors; the input text was characters from the Unicode Standard encoded in UTF-8.

The fonts used were Lucida Sans, in a special version incorporating over 1700 characters from the Unicode Standard, along with Lucida Sans Italic, Lucida Sans DemiBold, and Lucida Typewriter, designed by Bigelow & Holmes, Atherton, California. The hinted Adobe Type 1 representation of the fonts was provided by Y&Y Inc., 45 Walden Street, Concord, MA, 01742, USA.