Linux Manual Pages

Last updated on May 9, 1999.

NOTE: These are the man pages written by the Linux Documentation Project. Man pages for applications are not yet included here.

- Section 1
- Section 2
- Section 3
- Section 4
- Section 5
- Section 6
- Section 7
- Section 8
- Section 9

Linux Man Pages Section 1

- chgrp.1
- chmod.1
- chown.1
- cp.1
- dd.1
- df.1
- diff.1
- dircolors.1
- du.1
- install.1
- intro.1
- ln.1
- ls.1
- mkdir.1
- mkfifo.1
- mknod.1
- mv.1
- rm.1
- rmdir.1
- touch.1

NAME

chgrp - change group ownership of files

SYNOPSIS

chgrp [*options*] *group file*...

POSIX options: **[-R]**

GNU options (shortest form): **[-cfvR]** **[--help]** **[--version]** **[--]**

DESCRIPTION

chgrp changes the group ownership of each given *file* to *group*, which can be either a group name or a numeric group ID.

POSIX OPTIONS

-R Recursively change the group ownership of directories and their contents. (And continue even when errors are encountered.)

GNU OPTIONS

-c, --changes
Verbosely describe the action for each *file* whose group actually changes.

-f, --silent, --quiet
Do not print error messages about files whose group cannot be changed.

-h, --no-dereference
Act on symbolic links themselves instead of what they point to. Only available if the **lchown** system call is provided.

-v, --verbose
Verbosely describe the action or non-action taken for every *file*.

-R, --recursive
Recursively change the group ownership of directories and their contents.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2 only requires the -R option. Use of other options may not be portable.

NOTES

This page describes **chgrp** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl, aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org. Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

chmod - change access permissions of files

SYNOPSIS

chmod [*options*] *mode file*...

POSIX options: **[-R]**

GNU options (shortest form): **[-cfvR] [--help] [--version] [--]**

DESCRIPTION

chmod changes the permissions of each given *file* according to *mode*, which can be either a symbolic representation of changes to make, or an octal number representing the bit pattern for the new permissions.

The format of a symbolic mode change argument is `[ugoa...][[+-=][rwxXstugo...]...][,...]'.

Such an argument is a list of symbolic mode change commands, separated by commas. Each symbolic mode change command starts with zero or more of the letters `ugoa'; these control which users' access to the file will be changed: the user who owns it (u), other users in the file's group (g), other users not in the file's group (o), or all users (a). Thus, `a' is here equivalent to `ugo'. If none of these are given, the effect is as if `a' were given, but bits that are set in the umask are not affected.

The operator `+' causes the permissions selected to be added

to the existing permissions of each file; `-' causes them to be removed; and `=' causes them to be the only permissions that the file has.

The letters `rwxXstugo' select the new permissions for the affected users: read (r), write (w), execute (or access for directories) (x), execute only if the file is a directory or already has execute permission for some user (X), set user or group ID on execution (s), sticky bit (t), the permissions that the user who owns the file currently has for it (u), the permissions that other users in the file's group have for it (g), and the permissions that other users not in the file's group have for it (o). (Thus, `chmod g-s file' removes the set-group-ID (sgid) bit, `chmod ug+s file' sets both the suid and sgid bits, while `chmod o+s file' does nothing.)

The `sticky bit' is not described by POSIX. The name derives from the original meaning: keep program text on swap device. These days, when set for a directory, it means that only the owner of the file and the owner of that directory may remove the file from that directory. (This is commonly used on directories like /tmp that have general write permission.)

A numeric mode is from one to four octal digits (0-7), derived by adding up the bits with values 4, 2, and 1. Any omitted digits are assumed to be leading zeros. The first digit selects the set user ID (4) and set group ID (2) and save text image (1) attributes. The second digit selects permissions for the user who owns the file: read (4), write (2), and execute (1); the third selects permissions for other users in the file's group, with the same values; and the fourth for other users not in the file's group, with the same values.

chmod never changes the permissions of symbolic links, since the **chmod** system call cannot change their permissions. This is not a problem since the permissions of symbolic links are never used. However, for each symbolic link listed on the command line, **chmod** changes the permissions of the pointed-to file. In contrast, **chmod** ignores symbolic links encountered during recursive directory traversals.

POSIX OPTIONS

-R Recursively change permissions of directories and their contents.

GNU OPTIONS

-c, --changes
Verbosely describe the action for each *file* whose permissions actually changes.

-f, --silent, --quiet
Do not print error messages about files whose permissions cannot bechanged.

-v, --verbose
Verbosely describe the action or non-action taken for every *file*.

-R, --recursive
Recursively change permissions of directories and their contents.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2 only requires the -R option. Use of other options may not be portable. This standard does not describe the 't' permission bit. This standard does not specify whether **chmod** must preserve consistency by clearing or refusing to set the suid and sgid bits, e.g., when all execute bits are cleared, or whether **chmod** honors the `s' bit at all.

NONSTANDARD MODES

Above we described the use of the `t' bit on directories. Various systems attach special meanings to otherwise meaningless combinations of mode bits. In particular, Linux, following System V (see System V Interface Definition (SVID) Version 3), lets the sgid bit for files without group execute permission mark the file for mandatory locking. For more details, see the file */usr/src/linux/Documentation/mandatory.txt*.

NOTES

This page describes **chmod** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl, aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org. Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

chown - change file owner and group

SYNOPSIS

chown [*options***] *user*[:*group***

POSIX options: **[-R]**

GNU options (shortest form): **[-cfhvR] [--help] [--version] [--]**

DESCRIPTION

chown changes the user and/or group ownership of each given *file* as specified by the first non-option argument as follows: if only a user name (or numeric user ID) is given, that user is made the owner of each given file, and the files' group is not changed. If the user name is followed by a colon and a group name (or numeric group ID), with no spaces between them, the group ownership of the files is changed as well.

GNU DETAILS

The GNU version allows a dot instead of a colon (following BSD). [This was not allowed by POSIX since a dot is a valid character in a user name.] If a colon or dot but no group name follows the user name, that user is made the owner of

the files and the group of the files is changed to that user's login group. If the colon or dot and group are given, but the user name is omitted, only the group of the files is changed; in this case, **chown** performs the same function as **chgrp**.

POSIX OPTIONS

-R Recursively change ownership of directories and their contents.

GNU OPTIONS

-c, --changes
Verbosely describe the action for each *file* whose ownership actually changes.

-f, --silent, --quiet
Do not print error messages about files whose ownership cannot be changed.

-h, --no-dereference
Act on symbolic links themselves instead of what they point to. Only available if the **lchown** system call is provided.

-v, --verbose
Verbosely describe the action (or non-action) taken for every *file*.

-R, --recursive
Recursively change ownership of directories and their contents.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2 does not allow use of the dot as separator between user name and group name.

NOTES

This page describes **chown** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

cp - copy files and directories

SYNOPSIS

cp [*options*] *file path*
cp [*options*] *file... directory*

POSIX options: **[-fipRr]**

GNU options (shortest form): **[-abdfilprsuvxPR] [-S** *SUFFIX***] [-V {numbered,existing,simple}] [--sparse=***WHEN***] [--help] [--version] [--]**

DESCRIPTION

cp copies files (or, optionally, directories). You can either copy one file to a given destination, or copy arbitrarily many files to a destination directory.

If the last argument names an existing directory, **cp** copies each source *file* into that directory (retaining the same name). Otherwise, if only two files are given, it copies the first onto the second. It is an error if the last argument is not a directory and more than two non-option arguments are given.

(Thus, `cp -r /a /b' will copy /a to /b/a and /a/x to /b/a/x in case /b exists already, but it will copy /a to /b and /a/x to /b/x if there was no /b beforehand.)

The modes of the files and directories created will be the same as those of the original files, ANDed by 0777, and

modified by the user's umask (unless the -p option was specified). (But during the recursive copy of directories, newly created directories will temporarily get their final mode ORed with S_IRWXU (0700), so as to allow the process to read, write and search the newly created directory.)

Nothing is done when copying a file to itself (except possibly producing an error message). When copying to a different existing file, it is opened using `open(path, O_WRONLY | O_TRUNC)'. When copying to a new file it is created using `open(path, O_WRONLY | O_CREAT, mode)'. If this fails, the file existed, and the -f option was given, **cp** tries to delete (unlink) the existing file, and if this succeeds proceeds as for a new file.

POSIX OPTIONS

POSIX recognizes four options and a half:

-f Remove existing destination files if required. (See above.)

-i Prompt whether to overwrite existing regular destination files. (Write a question on stderr, and read the answer from stdin. Only copy upon an affirmative answer.)

-p Preserve the original files' owner, group, permissions (including the setuid and setgid bits), time of last modification and time of last access. In case duplication of owner or group fails, the setuid and setgid bits are cleared. (Note that afterwards source and copy may well have different times of last access, since the copy operation is an access to the source file.)

-R Copy directories recursively, and do the right thing when objects other than ordinary files or directories are encountered. (Thus, the copy of a FIFO or special file is a FIFO or special file.)

-r Copy directories recursively, and do something unspecified with objects other than ordinary files or directories. (Thus, it is allowed, in fact encouraged, to have the -r option a synonym for -R. However, silly behaviour, like that of the present GNU version of **cp** (see below) is not forbidden.)

GNU DETAILS

Generally, files are written just as they are read. For exceptions, see the --**sparse** option below.

By default, `cp' does not copy directories (see -**r** below).

cp generally refuses to copy a file onto itself, with the following exception: if --**force** --**backup** is specified with *source* and *dest* identical, and referring to a regular file, **cp** will make a backup file, either regular or numbered, as specified in the usual ways. This is useful when you simply want to make a backup of an existing file before changing it.

GNU OPTIONS

-a, --archive
Preserve as much as possible of the structure and attributes of the original files in the copy (but do not preserve directory structure). Equivalent to -**dpR**.

-d, --no-dereference
Copy symbolic links as symbolic links rather than copying the files that they point to, and preserve hard links between source files in the copies.

-f, --force
Remove existing destination files.

-i, --interactive

Prompt whether to overwrite existing regular destination files.

-l, --link
Make hard links instead of copies of non-directories.

-p, --preserve
Preserve the original files' owner, group, permissions, and timestamps.

-P, --parents
Form the name of each destination file by appending to the target directory a slash and the specified name of the source file. The last argument given to **cp** must be the name of an existing directory. For example, the command:

```
cp --parents a/b/c existing_dir
```

copies the file `a/b/c' to `existing_dir/a/b/c', creating any missing intermediate directories.

-r Copy directories recursively, copying any non-directories and non-symbolic links (that is, FIFOs and special files) as if they were regular files. This means trying to read the data in each source file and writing it to the destination. Thus, with this option, `cp' may well hang indefinitely reading a FIFO or /dev/tty. (This is a bug. It means that you have to avoid -r and use -R if you don't know what is in the tree you are copying. Opening an unknown device file, say a scanner, has unknown effects on the hardware.)

-R, --recursive
Copy directories recursively, preserving non-directories (see **-r** just above).

--sparse=*WHEN*
A `sparse file' contains `holes' - sequences of zero bytes that do not occupy any physical disk blocks; the `read' system call reads these as zeroes. This can both save considerable disk space and increase speed, since many binary files contain lots of consecutive zero bytes. By default, `cp' detects holes in input source files via a crude heuristic and makes the corresponding output file sparse as well.

The *WHEN* value can be one of the following:

auto The default behavior: the output file is sparse if the input file is sparse.

always
Always make the output file sparse. This is useful when the input file resides on a filesystem that does not support sparse files, but the output file is on a filesystem that does.

never
Never make the output file sparse. If you find an application for this option, let us know.

-s, --symbolic-link
Make symbolic links instead of copies of non-directories. All source file names must be absolute (starting with `/') unless the destination files are in the current directory. This option merely results in an error message on systems that do not support symbolic links.

-u, --update
Do not copy a nondirectory that has an existing destination with the same or newer modification time.

-v, --verbose
Print the name of each file before copying it.

-x, --one-file-system
Skip subdirectories that are on different filesystems from the one that the copy started on.

GNU BACKUP OPTIONS

The GNU versions of programs like **cp**, **mv**, **ln**, **install** and **patch** will make a backup of files about to be overwritten, changed or destroyed if that is desired. That backup files are desired is indicated by the -b option. How they should be named is specified by the -V option. In case the name of the backup file is given by the name of the file extended by a suffix, this suffix is specified by the -S option.

-b, --backup
Make backups of files that are about to be overwritten or removed.

-S *SUFFIX*, **--suffix=***SUFFIX*
Append *SUFFIX* to each backup file made. If this option is not specified, the value of the **SIMPLE_BACKUP_SUFFIX** environment variable is used. And if **SIMPLE_BACKUP_SUFFIX** is not set, the default is `~'.

-V *METHOD*, **--version-control=***METHOD*
Specify how backup files are named. The *METHOD* argument can be `numbered' (or `t'), `existing' (or `nil'), or `never' (or `simple'). If this option is not specified, the value of the **VERSION_CONTROL** environment variable is used. And if **VERSION_CONTROL** is not set, the default backup type is `existing'.

This option corresponds to the Emacs variable `version-control'. The valid *METHOD*s are (unique abbreviations are accepted):

t, **numbered**
Always make numbered backups.

nil, **existing**
Make numbered backups of files that already have them, simple backups of the others.

never, **simple**
Always make simple backups.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_COLLATE, LC_CTYPE and LC_MESSAGES have the usual meaning. For the GNU version, the variables SIMPLE_BACKUP_SUFFIX and VERSION_CONTROL control backup file naming, as described above.

CONFORMING TO

POSIX 1003.2

NOTES

This page describes **cp** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.

NAME

dd - convert and copy a file

SYNOPSIS

dd [--help] [--version] [if=*file***] [of=***file***] [ibs=***bytes***] [obs=***bytes***] [bs=***bytes***] [cbs=***bytes***] [skip=***blocks***] [seek=***blocks***] [count=***blocks***] [conv={ascii, ebcdic, ibm, block, unblock, lcase, ucase, swab, noerror, notrunc, sync}]**

DESCRIPTION

dd copies a file (from standard input to standard output, by default) using specific input and output blocksizes, while optionally performing conversions on it.

It reads the input one block at a time, using the specified input block size (the default is 512 bytes). If the **bs=***bytes* option was given, and no conversion other than **sync**, **noerror**, or **notrunc** was specified, it writes the amount of data read (which could be smaller than what was requested) in a separate output block. This output block has precisely the same length as was read unless the **sync** conversion was specified, in which case the data is padded with NULs (or spaces, see below).

Otherwise, the input, read one block at a time, is processed and the resulting output is collected and written in blocks of the specified output block size. The final output block may be shorter.

The numeric-valued options below (bytes and blocks) can be followed by a multiplier: `k'=1024, `b'=512, `w'=2, `c'=1

(`w' and `c' are GNU extensions; `w' should never be used - it means 2 in System V and 4 in 4.2BSD). Two or more of such numeric expressions can be multiplied by putting `x' in between.

OPTIONS

if=*file*
Read from *file* instead of standard input.

of=*file*
Write to *file* instead of standard output. Unless **conv=notrunc** is given, **dd** truncates *file* to zero bytes (or the size specified with **seek=**).

ibs=*bytes*
Read *bytes* bytes at a time. The default is 512.

obs=*bytes*
Write *bytes* bytes at a time. The default is 512.

bs=*bytes*
Both read and write *bytes* bytes at a time. This overrides **ibs** and **obs**. (And setting **bs** is not equivalent with setting both **ibs** and **obs** to this same value, at least when no conversion other than **sync**, **noerror** and **notrunc** is specified, since it stipulates that each input block shall be copied to the output as a single block without aggregating short blocks.)

cbs=*bytes*
Specify the conversion block size for **block** and **unblock**.

skip=*blocks*
Skip *blocks* **ibs**-byte blocks in the input file before copying.

seek=*blocks*
Skip *blocks* **obs**-byte blocks in the output file before copying.

count=*blocks*
Copy *blocks* **ibs**-byte blocks from the input file, instead of everything until the end of the file.

conv=*CONVERSION***[,***CONVERSION***]...**
Convert the file as specified by the *CONVERSION* argument(s). (No spaces around any comma(s).)

Conversions:

ascii
Convert EBCDIC to ASCII.

ebcdic
Convert ASCII to EBCDIC.

ibm Convert ASCII to alternate EBCDIC.

block
For each line in the input, output **cbs** bytes, replacing the input newline with a space and padding with spaces as necessary.

unblock
Replace trailing spaces in each **cbs**-sized input block with a newline.

lcase
Change uppercase letters to lowercase.

ucase
Change lowercase letters to uppercase.

swab Swap every pair of input bytes. GNU **dd**, unlike others, works when an odd number of bytes are read - the last byte is simply copied (since there is nothing to swap it with).

noerror
Continue after read errors.

notrunc
Do not truncate the output file.

sync Pad every input block to size of **ibs** with trailing zero bytes.

GNU STANDARD OPTIONS

--**help**
Print a usage message on standard output and exit successfully.

--**version**
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2

NOTES

This page describes **dd** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

df - report filesystem disk space usage

SYNOPSIS

df [*options*] [*file*...]

POSIX options: **[-kP]**

GNU options (shortest form): **[-ahikmPv] [-t** *fstype***] [-x** *fstype***] [--print-type] [--no-sync] [--sync] [--help] [--version] [--]**

DESCRIPTION

df reports the amount of disk space used and available on filesystems.

With no arguments, **df** reports the space used and available on all currently mounted filesystems (of all types). Otherwise, **df** reports on the filesystem containing each argument *file*.

POSIX DETAILS

The output is in 512-byte units by default, but in 1024-byte units when the -k option is given. The output format is undefined, unless the -P option is given. If *file* is not a regular file, a directory or a FIFO, the result is unspeci-

fied.

GNU DETAILS

The output is in 1024-byte units (when no units are specified by options), unless the environment variable **POSIXLY_CORRECT** is set, in which case POSIX is followed.

If an argument *file* is a disk device file containing a mounted filesystem, **df** shows the space available on that filesystem rather than on the filesystem containing the device node.

POSIX OPTIONS

-k Use 1024-byte units instead of the default 512-byte units.

-P Output in six columns, with heading `Filesystem N-blocks Used Available Capacity Mounted on' (with N=512, but N=1024 when the -k option is given).

GNU OPTIONS

-a, --all
Include in the listing filesystems that have a size of 0 blocks, which are omitted by default. Such filesystems are typically special-purpose pseudo-filesystems, such as automounter entries. Also, filesystems of type "ignore" or "auto", supported by some operating systems, are only included if this option is specified.

-h, --human-readable
Append a size letter such as **M** for megabytes to each size.

-i, --inodes
List inode usage information instead of block usage. An inode (short for index node) is contains information about a file such as its owner, permissions, timestamps, and location on the disk.

-k, --kilobytes
Print sizes in 1024-byte blocks.

-m, --megabytes
Print sizes in megabyte (that's 1,048,576 bytes) blocks.

--no-sync
Do not invoke the **sync** system call before getting any usage data. This may make **df** run significantly faster, but on some systems (notably SunOS) the results may be slightly out of date. This is the default.

-P, --portability
Use the POSIX output format. This is like the default format except that the information about each filesystem is always printed on exactly one line; a mount device is never put on a line by itself. This means that if the mount device name is more than 20 characters long (e.g., for some network mounts), the columns are misaligned.

--sync
Invoke the **sync** system call before getting any usage data. On some systems (notably SunOS), doing this yields more up to date results, but in general this option makes **df** much slower, especially when there are many or very busy filesystems.

-t *fstype*, **--type=***fstype*
Limit the listing to filesystems of type *fstype*. Multiple filesystem types can be specified by giving multiple **-t** options. By default, nothing is omitted.

-T, --print-type
Print each filesystem's type. The types given are those reported by the system (and are found in a system-dependent way, for example by reading */etc/mtab)*. See also **mount**(8).

-x *fstype*, **--exclude-type=***fstype*
Limit the listing to filesystems not of type *fstype*. Multiple filesystem types can be eliminated by giving multiple **-x** options. By default, no filesystem types are omitted.

-v Ignored; for compatibility with System V versions of **df**.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variable POSIXLY_CORRECT determines the choice of unit. If it is not set, and the variable BLOCKSIZE has a value starting with `HUMAN', then behaviour is as for the -h option, unless overridden by -k or -m options. The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2

SEE ALSO

mount(8)

NOTES

This page describes **df** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.

NAME

diff - find differences between two files

SYNOPSIS

diff [options] from-file to-file

DESCRIPTION

In the simplest case, *diff* compares the contents of the two files *from-file* and *to-file*. A file name of - stands for text read from the standard input. As a special case, **diff** - - compares a copy of standard input to itself.

If *from-file* is a directory and *to-file* is not, *diff* compares the file in *from-file* whose file name is that of *to-file*, and vice versa. The non-directory file must not be -.

If both *from-file* and *to-file* are directories, *diff* compares corresponding files in both directories, in alphabetical order; this comparison is not recursive unless the **-r** or **--recursive** option is given. *diff* never compares the actual contents of a directory as if it were a file. The file that is fully specified may not be standard input, because standard input is nameless and the notion of ``file with the same name'' does not apply.

diff options begin with -, so normally *from-file* and *to-file* may not begin with -. However, -- as an argument by itself treats the remaining arguments as file names even if they begin with -.

Options

Below is a summary of all of the options that GNU *diff* accepts. Most options have two equivalent names, one of which is a single letter preceded by -, and the other of which is a long name preceded by --. Multiple single letter options (unless they take an argument) can be combined into a single command line word: **-ac** is equivalent to **-a -c**. Long named options can be abbreviated to any unique prefix of their name. Brackets (**[** and **]**) indicate that an option takes an optional argument.

-*lines*
Show *lines* (an integer) lines of context. This option does not specify an output format by itself; it has no effect unless it is combined with **-c** or **-u**. This option is obsolete. For proper operation, *patch* typically needs at least two lines of context.

-a Treat all files as text and compare them line-by-line, even if they do not seem to be text.

-b Ignore changes in amount of white space.
-B Ignore changes that just insert or delete blank lines.

--**brief**
Report only whether the files differ, not the details of the differences.

-c Use the context output format.

-C *lines*
--**context[=***lines***]**
Use the context output format, showing *lines* (an integer) lines of context, or three if *lines* is not given. For proper operation, *patch* typically needs at least two lines of context.

--**changed-group-format=***format*
Use *format* to output a line group containing differing lines from both files in if-then-else format.

-d Change the algorithm to perhaps find a smaller set of changes. This makes *diff* slower (sometimes much slower).

-D *name*
Make merged if-then-else format output, conditional on

the preprocessor macro *name*.

-e
--ed Make output that is a valid *ed* script.

--exclude=*pattern*
When comparing directories, ignore files and subdirectories whose basenames match *pattern*.

--exclude-from=*file*
When comparing directories, ignore files and subdirectories whose basenames match any pattern contained in *file*.

--expand-tabs
Expand tabs to spaces in the output, to preserve the alignment of tabs in the input files.

-f Make output that looks vaguely like an *ed* script but has changes in the order they appear in the file.

-F *regexp*
In context and unified format, for each hunk of differences, show some of the last preceding line that matches *regexp*.

--forward-ed
Make output that looks vaguely like an **ed** script but has changes in the order they appear in the file.

-h This option currently has no effect; it is present for Unix compatibility.

-H Use heuristics to speed handling of large files that have numerous scattered small changes.

--horizon-lines=*lines*
Do not discard the last *lines* lines of the common prefix and the first *lines* lines of the common suffix.

-i Ignore changes in case; consider upper- and lower-case letters equivalent.

-I *regexp*
Ignore changes that just insert or delete lines that match *regexp*.

--ifdef=*name*
Make merged if-then-else format output, conditional on the preprocessor macro *name*.

--ignore-all-space
Ignore white space when comparing lines.

--ignore-blank-lines
Ignore changes that just insert or delete blank lines.

--ignore-case
Ignore changes in case; consider upper- and lower-case to be the same.

--ignore-matching-lines=*regexp*
Ignore changes that just insert or delete lines that match *regexp*.

--ignore-space-change
Ignore changes in amount of white space.

--initial-tab
Output a tab rather than a space before the text of a line in normal or context format. This causes the alignment of tabs in the line to look normal.

-l
Pass the output through *pr* to paginate it.

-L *label*
--label=*label*
Use *label* instead of the file name in the context format and unified format headers.

--left-column
Print only the left column of two common lines in side by side format.

--line-format=*format*
Use *format* to output all input lines in in-then-else format.

--minimal
Change the algorithm to perhaps find a smaller set of changes. This makes *diff* slower (sometimes much slower).

-n Output RCS-format diffs; like **-f** except that each command specifies the number of lines affected.

-N
--new-file
In directory comparison, if a file is found in only one directory, treat it as present but empty in the other directory.

--new-group-format=*format*
Use *format* to output a group of lines taken from just the second file in if-then-else format.

--new-line-format=*format*
Use *format* to output a line taken from just the second file in if-then-else format.

--old-group-format=*format*
Use *format* to output a group of lines taken from just the first file in if-then-else format.

--old-line-format=*format*
Use *format* to output a line taken from just the first file in if-then-else format.

-p Show which C function each change is in.

-P When comparing directories, if a file appears only in the second directory of the two, treat it as present but empty in the other.

--paginate
Pass the output through *pr* to paginate it.

-q Report only whether the files differ, not the details of the differences.

-r When comparing directories, recursively compare any subdirectories found.

--rcs
Output RCS-format diffs; like **-f** except that each command specifies the number of lines affected.

--recursive

When comparing directories, recursively compare any subdirectories found.

--report-identical-files
-s Report when two files are the same.

-S *file*
When comparing directories, start with the file *file*. This is used for resuming an aborted comparison.

--sdiff-merge-assist
Print extra information to help *sdiff*. *sdiff* uses this option when it runs *diff*. This option is not intended for users to use directly.

--show-c-function
Show which C function each change is in.

--show-function-line=*regexp*
In context and unified format, for each hunk of differences, show some of the last preceding line that matches *regexp*.

--side-by-side
Use the side by side output format.

--speed-large-files
Use heuristics to speed handling of large files that have numerous scattered small changes.

--starting-file=*file*
When comparing directories, start with the file *file*. This is used for resuming an aborted comparison.

--suppress-common-lines
Do not print common lines in side by side format.

-t Expand tabs to spaces in the output, to preserve the alignment of tabs in the input files.

-T Output a tab rather than a space before the text of a line in normal or context format. This causes the alignment of tabs in the line to look normal.

--text
Treat all files as text and compare them line-by-line,

even if they do not appear to be text.

-u Use the unified output format.

--unchanged-group-format=*format*
Use *format* to output a group of common lines taken from both files in if-then-else format.

--unchanged-line-format=*format*
Use *format* to output a line common to both files in if-then-else format.

--unidirectional-new-file
When comparing directories, if a file appears only in the second directory of the two, treat it as present but empty in the other.

-U *lines*
--unified[=*lines***]**
Use the unified output format, showing *lines* (an integer) lines of context, or three if *lines* is not given. For proper operation, *patch* typically needs at least two lines of context.

-v
--version
Output the version number of *diff*.

-w Ignore white space when comparing lines.

-W *columns*
--width=*columns*
Use an output width of *columns* in side by side format.

-x *pattern*
When comparing directories, ignore files and subdirectories whose basenames match *pattern*.

-X *file*
When comparing directories, ignore files and subdirectories whose basenames match any pattern contained in *file*.

-y Use the side by side output format.

SEE ALSO

cmp(1), comm(1), diff3(1), ed(1), patch(1), pr(1), sdiff(1).

DIAGNOSTICS

An exit status of 0 means no differences were found, 1 means some differences were found, and 2 means trouble.

NAME

dircolors - color setup for `ls'

SYNOPSIS

dircolors [-b] [--sh] [--bourne-shell] [-c] [--csh] [--c-shell] [-p] [--print-database] [--help] [--version] [*FILE*]

DESCRIPTION

dircolors outputs a sequence of shell commands to define the desired color output from **ls** (and **dir**, etc.). Typical usage:

```
eval `dircolors [OPTION]... [FILE]`
```

If *FILE* is specified, **dircolors** reads it to determine which colors to use for which file types and extensions. Otherwise, a precompiled database is used. For details on the format of these files, run `dircolors --print-database'.

The output is a shell command to set the **LS_COLORS** environment variable. You can specify the shell syntax to use on the command line, or **dircolors** will guess it from the value of the **SHELL** environment variable.

After execution of this command, `ls --color' (which one might alias to ls) will list files in the desired colors.

OPTIONS

-b, --sh, --bourne-shell
Output Bourne shell commands. This is the default if the **SHELL** environment variable is set and does not end with *csh* or *tcsh*.

-c, --csh, --c-shell
Output C shell commands. This is the default if **SHELL** ends with *csh* or *tcsh*.

-p, --print-database
Print the (compiled-in) default color configuration database. This output is itself a valid configuration file, and is fairly descriptive of the possibilities.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables SHELL and TERM are used to find the proper form of the shell command. The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning. The variable LS_COLORS is used to transfer information to **ls**.

CONFORMING TO

Coloured output for **ls**(1) is a GNU extension.

SEE ALSO

ls(1)

NOTES

This page describes **dircolors** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

du - estimate file space usage

SYNOPSIS

du [*options*] [*file*...]

POSIX options: **[-askx]**

GNU options (shortest form): **[-abchklmsxDLS]** **[--help]** **[--version]** **[--]**

DESCRIPTION

du reports the amount of disk space used by the specified files, and by each directory in the hierarchies rooted at the specified files. Here `disk space used' means space used for the entire file hierarchy below the specified file.

With no arguments, **du** reports the disk space for the current directory.

POSIX DETAILS

The output is in 512-byte units by default, but in 1024-byte units when the -k option is given.

GNU DETAILS

The output is in 1024-byte units (when no units are specified by options), unless the environment variable **POSIXLY_CORRECT** is set, in which case POSIX is followed.

POSIX OPTIONS

-a Show counts for all files encountered, not just directories.

-k Use 1024-byte units instead of the default 512-byte units.

-s Only output space usage for the actual arguments given, not for their subdirectories.

-x Only count space on the same device as the argument given.

GNU OPTIONS

-a, --all
Show counts for all files, not just directories.

-b, --bytes
Print sizes in bytes, instead of kilobytes.

-c, --total
Print a grand total of all arguments after all arguments have been processed. This can be used to find out the total disk usage of a given set of files or directories.

-D, --dereference-args
Dereference symbolic links that are command line arguments. Does not affect other symbolic links. This is

helpful for finding out the disk usage of directories, such as */usr/tmp*, which are often symbolic links.

-h, --human-readable
Append a size letter, such as **M** for megabytes, to each size.

-k, --kilobytes
Print sizes in kilobytes.

-l, --count-links
Count the size of all files, even if they have appeared already (as a hard link).

-L, --dereference
Dereference symbolic links (show the disk space used by the file or directory that the link points to instead of the space used by the link).

-m, --megabytes
Print sizes in megabyte (that 1,048,576 bytes) blocks.

-s, --summarize
Display only a total for each argument.

-S, --separate-dirs
Report the size of each directory separately, not including the sizes of subdirectories.

-x, --one-file-system
Skip directories that are on different filesystems from the one that the argument being processed is on.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

BUGS

On BSD systems, **du** reports sizes that are half the correct values for files that are NFS-mounted from HP-UX systems. On HP-UX systems, it reports sizes that are twice the correct values for files that are NFS-mounted from BSD systems. This is due to a flaw in HP-UX; it also affects the HP-UX **du** program.

ENVIRONMENT

The variable POSIXLY_CORRECT determines the choice of unit. If it is not set, and the variable BLOCKSIZE has a value starting with `HUMAN', then behaviour is as for the -h option, unless overridden by -k or -m options. The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2

NOTES

This page describes **du** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to

fileutils-bugs@gnu.ai.mit.edu.

NAME

install - copy files and set attributes

SYNOPSIS

install [*options*] **[-s] [--strip]** *source dest*
install [*options*] **[-s] [--strip]** *source... directory*
install [*options*] **[-d,--directory]** *directory...*

Options (shortest form):
[-b] [-c] [-g *group***] [-m** *mode***] [-o** *owner***] [-S** *SUFFIX***] [-V {numbered,existing,simple}] [--help] [--version] [--]**

DESCRIPTION

install copies files while setting their permission modes and, if possible, their owner and group.

In the first of these invocation forms, the *source* file is copied to the *dest* target file. In the second, each of the *source* files are copied to the destination *directory*. In the last, each *directory* (and any missing parent directories) is created.

install is similar to **cp**, but allows you to control the attributes of destination files. It is typically used in Makefiles to copy programs into their destination directories. It refuses to copy files onto themselves.

OPTIONS

-c Ignored; for compatibility with old Unix versions of **install**.

-d, --directory
Create each given directory and any missing parent directories, setting the owner, group and mode as given on the command line or to the defaults. It also gives any parent directories it creates those attributes. (This is different from the SunOS 4.x **install**, which gives directories that it creates the default attributes.)

-g *group*, **--group=***group*
Set the group ownership of installed files or directories to *group*. The default is the process's current group. *group* may be either a group name or a numeric group id.

-m *mode*, **--mode=***mode*
Set the permissions for the installed file or directory to *mode*, which can be either an octal number, or a symbolic mode as in **chmod**, with 0 as the point of departure. The default mode is 0755 - read, write, and execute for the owner, and read and execute for group and other.

-o *owner*, **--owner=***owner*
If **install** has appropriate privileges (is run as root), set the ownership of installed files or directories to *owner*. The default is `root'. *owner* may be either a user name or a numeric user ID.

-s, --strip
Strip the symbol tables from installed binary executables.

GNU BACKUP OPTIONS

The GNU versions of programs like **cp**, **mv**, **ln**, **install** and **patch** will make a backup of files about to be overwritten, changed or destroyed if that is desired. That backup files are desired is indicated by the -b option. How they should be named is specified by the -V option. In case the name of the backup file is given by the name of the file extended by a suffix, this suffix is specified by the -S option.

-b, **--backup**
Make backups of files that are about to be overwritten or removed.

-S *SUFFIX*, **--suffix=***SUFFIX*
Append *SUFFIX* to each backup file made. If this option is not specified, the value of the **SIMPLE_BACKUP_SUFFIX** environment variable is used. And if **SIMPLE_BACKUP_SUFFIX** is not set, the default is `~'.

-V *METHOD*, **--version-control=***METHOD*
Specify how backup files are named. The *METHOD* argument can be `numbered' (or `t'), `existing' (or `nil'), or `never' (or `simple'). If this option is not specified, the value of the **VERSION_CONTROL** environment variable is used. And if **VERSION_CONTROL** is not set, the default backup type is `existing'.

This option corresponds to the Emacs variable `version-control'. The valid *METHOD*s are (unique abbreviations are accepted):

t, **numbered**
Always make numbered backups.

nil, **existing**
Make numbered backups of files that already have them, simple backups of the others.

never, **simple**
Always make simple backups.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning. For the GNU version, the variables SIMPLE_BACKUP_SUFFIX and VERSION_CONTROL control backup file naming, as described above.

CONFORMING TO

BSD 4.2 (which had the -c, -m, -o, -g and -s options).

NOTES

This page describes **install** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

intro - Introduction to user commands

DESCRIPTION

This chapter describes user commands.

AUTHORS

Look at the header of the manual page for the author(s) and copyright conditions. Note that these can be different from page to page!


TOC

NAME

ln - make links between files

SYNOPSIS

ln [options] *source* **[***dest***]**
ln [options] *source*... *directory*

POSIX options: **[-f]**

GNU options (shortest form): **[-bdfinsvF] [-S** *backup-suffix***] [-V {numbered,existing,simple}] [--help] [--version] [--]**

DESCRIPTION

There are two concepts of `link' in Unix, usually called hard link and soft link. A hard link is just a name for a file. (And a file can have several names. It is deleted from disk only when the last name is removed. The number of names is given by **ls**(1). There is no such thing as an `original' name: all names have the same status. Usually, but not necessarily, all names of a file are found in the filesystem that also contains its data.)

A soft link (or symbolic link, or symlink) is an entirely different animal: it is a small special file that contains a pathname. Thus, soft links can point at files on different filesystems (possibly NFS mounted from different machines), and need not point to actually existing files. When accessed (with the **open**(2) or **stat**(2) system calls), a reference to a symlink is replaced by the operating system kernel with a reference to the file named by the path name. (However, with **rm**(1) and **unlink**(2) the link itself is

removed, not the file it points to. There are special system calls **lstat**(2) and **readlink**(2) that read the status of a symlink and the filename it points to. For various other system calls there is some uncertainty and variation between operating systems as to whether the operation acts on the symlink itself, or on the file pointed to.)

ln makes links between files. By default, it makes hard links; with the **-s** option, it makes symbolic (or `soft') links.

If only one file is given, it links that file into the current directory, that is, creates a link to that file in the current directory, with name equal to (the last component of) the name of that file. (This is a GNU extension.) Otherwise, if the last argument names an existing directory, **ln** will create links to each mentioned *source* file in that directory, with a name equal to (the last component of) the name of that *source* file. (But see the description of the --**no**-**dereference** option below.) Otherwise, if only two files are given, it creates a link named *dest* to the file *source*. It is an error if the last argument is not a directory and more than two files are given.

By default, **ln** does not remove existing files or existing symbolic links. (Thus, it can be used for locking purposes: it will succeed only if *dest* did not exist already.) But it can be forced to do so with the option -f.

On existing implementations, if it is at all possible to make a hard link to a directory, this may be done by the superuser only. POSIX forbids the system call **link**(2) and the utility **ln** to make hard links to directories (but does not forbid hard links to cross filesystem boundaries).

POSIX OPTIONS

-f Remove existing destination files.

GNU OPTIONS

-d, -F, --directory
Allow the super-user to make hard links to directories.

-f, --force
Remove existing destination files.

-i, --interactive
Prompt whether to remove existing destination files.

-n, --no-dereference
When given an explicit destination that is a symlink to a directory, treat that destination as if it were a normal file.
When the destination is an actual directory (not a symlink to one), there is no ambiguity. The link is created in that directory. But when the specified destination is a symlink to a directory, there are two ways to treat the user's request. **ln** can treat the destination just as it would a normal directory and create the link in it. On the other hand, the destination can be viewed as a non-directory - as the symlink itself. In that case, **ln** must delete or backup that symlink before creating the new link. The default is to treat a destination that is a symlink to a directory just like a directory.

-s, --symbolic
Make symbolic links instead of hard links. This option merely produces an error message on systems that do not support symbolic links.

-v, --verbose
Print the name of each file before linking it.

GNU BACKUP OPTIONS

The GNU versions of programs like **cp**, **mv**, **ln**, **install** and **patch** will make a backup of files about to be overwritten,

changed or destroyed if that is desired. That backup files are desired is indicated by the -b option. How they should be named is specified by the -V option. In case the name of the backup file is given by the name of the file extended by a suffix, this suffix is specified by the -S option.

-b, --backup
Make backups of files that are about to be overwritten or removed.

-S *SUFFIX*, **--suffix=***SUFFIX*
Append *SUFFIX* to each backup file made. If this option is not specified, the value of the **SIMPLE_BACKUP_SUFFIX** environment variable is used. And if **SIMPLE_BACKUP_SUFFIX** is not set, the default is `~'.

-V *METHOD*, **--version-control=***METHOD*
Specify how backup files are named. The *METHOD* argument can be `numbered' (or `t'), `existing' (or `nil'), or `never' (or `simple'). If this option is not specified, the value of the **VERSION_CONTROL** environment variable is used. And if **VERSION_CONTROL** is not set, the default backup type is `existing'.

This option corresponds to the Emacs variable `version-control'. The valid *METHOD*s are (unique abbreviations are accepted):

t, numbered
Always make numbered backups.

nil, existing
Make numbered backups of files that already have them, simple backups of the others.

never, simple
Always make simple backups.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit suc-

cessfully.

--**version**
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2. However, POSIX 1003.2 (1996) does not discuss soft links. Soft links were introduced by BSD, and do not occur in System V release 3 (and older) systems.

SEE ALSO

ls(1), **rm**(1), **link**(2), **lstat**(2), **open**(2), **readlink**(2), **stat**(2), **unlink**(2)

NOTES

This page describes **ln** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

ls, dir, vdir - list directory contents

SYNOPSIS

ls [*options*] [*file*...]

POSIX options: **[-CFRacdilqrtu1]**

GNU options (shortest form): **[-1abcdfgiklmnopqrstux-ABCDFGLNQRSUX]** **[-w** *cols***]** **[-T** *cols***]** **[-I** *pattern***]** **[--full-time]**
[--format={long,verbose,commas,across,vertical,single-column}]
[--sort={none,time,size,extension}]
[--time={atime,access,use,ctime,status}]
[--color[={none,auto,always}]] [--help] [--version] [--]

DESCRIPTION

The program **ls** lists first its non-directory *file* arguments, and then for each directory argument all listable files contained within that directory. If no non-option arguments are present, a default argument `.' (the current directory) is assumed. The -d option causes directories to be treated as non-directory arguments. A file is listable when either its name does not start with `.', or the -a option is given.

Each of the lists of files (that of non-directory files, and for each directory the list of files inside) is sorted separately according to the collating sequence in the current locale. When the -l option is given, each list is preceded by a summary line giving the total size of all files in the list, measured in semi-kilobytes (512 B).

The output is to stdout, one entry per line, unless multicolumn output is requested by the -C option. However, for output to a terminal, it is undefined whether the output will be single-column or multi-column. The options -1 and -C can be used to force single-column and multi-column output, respectively.

POSIX OPTIONS

-C List files in columns, sorted vertically.

-F Suffix each directory name with `/', each FIFO name with `|', and each name of an executable with `*'.

-R Recursively list subdirectories encountered.

-a Include files with a name starting with `.' in the listing.

-c Use the status change time instead of the modification time for sorting (with -t) or listing (with -l).

-d List names of directories like other files, rather than listing their contents.

-i Precede the output for the file by the file serial number (i-node number).

-l Write (in single-column format) the file mode, the number of links to the file, the owner name, the group name, the size of the file (in bytes), the timestamp, and the filename. By default, the timestamp shown is that of the last modification; the options -c and -u select the other two timestamps. For device special files the size field is commonly replaced by the major and minor device numbers.

-q Output nonprintable characters in a filename as question marks. (This is permitted to be the default for output to a terminal.)

-r Reverse the order of the sort.

-t Sort by the timestamp shown.

-u Use the time of last access instead of the modification time for sorting (with -t) or listing (with -l).

-1 For single-column output.

GNU DETAILS

If standard output is a terminal, the output is in columns (sorted vertically).

dir (also installed as **d**) is equivalent to `ls -C'; that is, files are by default listed in columns, sorted vertically. **vdir** (also installed as **v**) is equivalent to `ls -l'; that is, files are by default listed in long format.

GNU OPTIONS

-1, --format=single-column
List one file per line. This is the default for when standard output is not a terminal.

-a, --all
List all files in directories, including all files that start with `.'.

-b, --escape
Quote nongraphic characters in file names using alphabetic and octal backslash sequences like those used in C.

-c, --time=ctime, --time=status
Sort directory contents according to the files' status change time (the `ctime' in the inode). If the long listing format is being used (**-l**) print the status

change time instead of the modification time.

-d, --directory
List names of directories like other files, rather than listing their contents.

-f Do not sort directory contents; list them in whatever order they are stored on the disk. Also enables **-a** and disables **-l**, **--color**, and **-s** if they were specified before the **-f**.

-g Ignored; for Unix compatibility.

-i, --inode
Print the inode number (also called the file serial number and index number) of each file to the left of the file name. (This number uniquely identifies each file within a particular filesystem)

-k, --kilobytes
If file sizes are being listed, print them in kilobytes.

-l, --format=long, --format=verbose
In addition to the name of each file, print the file type, permissions, number of hard links, owner name, group name, size in bytes, and timestamp (the modification time unless other times are selected). For files with a time that is more than 6 months old or more than 1 hour into the future, the timestamp contains the year instead of the time of day.

For each directory that is listed, preface the files with a line `total *blocks*', where *blocks* is the total disk space used by all files in that directory. By default, 1024-byte blocks are used; if the environment variable **POSIXLY_CORRECT** is set, 512-byte blocks are used (unless the **-k** option is given). The *blocks* computed counts each hard link separately; this is arguably a deficiency.

The permissions listed are similar to symbolic mode specifications but **ls** combines multiple bits into the third character of each set of permissions

s If the setuid or setgid bit and the corresponding

executable bit are both set.

S If the setuid or setgid bit is set but the corresponding executable bit is not set.

t If the sticky bit and the other-executable bit are both set.

T If the sticky bit is set but the other-executable bit is not set.

x If the executable bit is set and none of the above apply.

- Otherwise.

-m, --format=commas
List files horizontally, with as many as will fit on each line, each separated by a comma and a space.

-n, --numeric-uid-gid
List the numeric UID and GID instead of the names.

-o Produce long format directory listings, but don't display group information. It is equivalent to using **--format=long --no-group**. This option is provided for compatibility with other versions of **ls**.

-p Append a character to each file name indicating the file type. This is like **-F** except that executables aren't marked.

-q, --hide-control-chars
Print question marks instead of nongraphic characters in file names. This is the default.

-r, --reverse
Sort directory contents in reverse order.

-s, --size
Print the size of each file in 1024-byte blocks to the left of the file name. If the environment variable **POSIXLY_CORRECT** is set, 512-byte blocks are used instead, unless the **-k** option is given.

-t, --sort=time

Sort by modification time (the `mtime' in the inode) instead of alphabetically, with the newest files listed first.

-u, --time=atime, --time=access, --time=use
Sort directory contents according to the files' last access time instead of the modification time (the `atime' in the inode). If the long listing format is being used, print the last access time instead of the modification time.

-w, --width *cols*
Assume the screen is *cols* columns wide. The default is taken from the terminal driver if possible; otherwise the environment variable **COLUMNS** is used if it is set; otherwise the default is 80.

-x, --format=across, --format=horizontal
List the files in columns, sorted horizontally.

-A, --almost-all
List all files in directories, except for `.' and `..'.

-B, --ignore-backups
Do not list files that end with `~', unless they are given on the command line.

-C, --format=vertical
List files in columns, sorted vertically. This is the default if standard output is a terminal. It is always the default for **dir** and **d**.

-D, --dired
With the long listing (**-l**) format, print an additional line after the main output:
//DIRED// *BEG1 END1 BEG2 END2* ...

The *BEGn* and *ENDn* are unsigned integers which record the byte position of the beginning and end of each file name in the output. This makes it easy for Emacs to find the names, even when they contain unusual characters such as space or newline, without fancy searching.

If directories are being listed recursively (**-R**), output a similar line after each subdirectory:
//SUBDIRED// *BEG1 END1* ...

-F, --classify, --file-type
Append a character to each file name indicating the file type. For regular files that are executable, append a `*'. The file type indicators are `/' for directories, `@' for symbolic links, `|' for FIFOs, `=' for sockets, and nothing for regular files.

-G, --no-group
Inhibit display of group information in a long format directory listing.

-I, --ignore*pattern*
Do not list files whose names match the shell pattern *pattern* (not regular expression) unless they are given on the command line. As in the shell, an initial `.' in a filename does not match a wildcard at the start of *pattern*.

-L, --dereference
List the file information corresponding to the referrents of symbolic links rather for the links themselves.

-N, --literal
Do not quote file names.

-Q, --quote-name
Enclose file names in double quotes and quote nongraphic characters as in C.

-R, --recursive
List the contents of all directories recursively.

-S, --sort=size
Sort directory contents by file size instead of alphabetically, with the largest files listed first.

-T, --tabsize *cols*
Assume that each tabstop is *cols* columns wide. The default is 8. **ls** uses tabs where possible in the output, for efficiency. If *cols* is zero, do not use tabs at all.

-U, --sort=none
Do not sort directory contents; list them in whatever

order they are stored on the disk. (The difference between -**U**and-**f** is that the later doesn't disable or enable options). This is especially useful when listing very large directories, since not doing any sorting can be noticeably faster.

-**X**, --**sort=extension**
Sort directory contents alphabetically by file extension (characters after the last `.'); files with no extension are sorted first.

--**color[=***when***]**
Specify whether to use color for distinguishing file types. Colors are specified using the LS_COLORS environment variable. For information on how to set this variable, see **dircolors**(1). *when* may be omitted, or one of:

none Do not use color at all. This is the default.

auto Only use color if standard output is a terminal.
always
Always use color. Specifying --**color** and no *when* is equivalent to --**color=always**.

--**full-time**
List times in full, rather than using the standard abbreviation heuristics. The format is the same as **date**(1)'s default; it's not possible to change this, but you can extract out the date string with **cut**(1) and then pass the result to `date -d'.

This is most useful because the time output includes the seconds. (Unix filesystems store file timestamps only to the nearest second, so this option shows all the information there is.) For example, this can help when you have a Makefile that is not regenerating files properly.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variable POSIXLY_CORRECT determines the choice of unit. If it is not set, then the variable TABSIZE determines the number of chars per tab stop. The variable COLUMNS (when it contains the representation of a decimal integer) determines the output column width (for use with the -C option). Filenames must not be truncated to make them fit a multi-column output. The variables LANG, LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES and LC_TIME have the usual meaning. The variable TZ gives the time zone for time strings written by **ls**. The variable LS_COLORS is used to specify the colors used.

BUGS

On BSD systems, the **-s** option reports sizes that are half the correct values for files that are NFS-mounted from HP-UX systems. On HP-UX systems, **ls** reports sizes that are twice the correct values for files that are NFS-mounted from BSD systems. This is due to a flaw in HP-UX; it also affects the HP-UX **ls** program.

CONFORMING TO

POSIX 1003.2

SEE ALSO

dircolors(1)

NOTES

This page describes **ls** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

mkdir - make directories

SYNOPSIS

mkdir [*options***]** *directory*...

POSIX options: **[-p] [-m** *mode***]**

GNU options (shortest form): **[-p] [-m** *mode***] [--verbose] [--help] [--version] [--]**

DESCRIPTION

mkdir creates directories with the specified names.

By default, the mode of created directories is 0777 (`a+rwx') minus the bits set in the umask.

OPTIONS

-m *mode*, **--mode=***mode*
Set the mode of created directories to *mode*, which may be symbolic as in **chmod**(1) and then uses the default mode as the point of departure.

-p, --parents
Make any missing parent directories for each *directory* argument. The mode for parent directories is set to

the umask modified by `u+wx'. Ignore arguments corresponding to existing directories. (Thus, if a directory /a exists, then `mkdir /a' is an error, but `mkdir -p /a' is not.)

--**verbose**
Print a message for each created directory. This is most useful with --**parents**.

GNU STANDARD OPTIONS

--**help**
Print a usage message on standard output and exit successfully.

--**version**
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2

NOTES

This page describes **mkdir** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

mkfifo - make FIFOs (named pipes)

SYNOPSIS

mkfifo [*options***]** *file*...

POSIX options: **[-m** *mode***]**

GNU options (shortest form): **[-m** *mode***] [--help] [--version] [--]**

DESCRIPTION

mkfifo creates FIFOs (also called "named pipes") with the specified filenames.

A "FIFO" is a special file type that permits independent processes to communicate. One process opens the FIFO file for writing, and another for reading, after which data can flow as with the usual anonymous pipe in shells or elsewhere.

By default, the mode of created FIFOs is 0666 (`a+rw') minus the bits set in the umask.

OPTIONS

-m *mode*, **--mode=***mode*
Set the mode of created FIFOs to *mode*, which is symbolic as in **chmod**(1) and uses the default mode as the point of departure.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2

NOTES

This page describes **mkfifo** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

mknod - make block or character special files

SYNOPSIS

mknod [*options*] *name* **{bc}** *major minor*
mknod [*options*] *name* **p**

GNU options (shortest form): **[-m** *mode***] [--help] [--version] [--]**

DESCRIPTION

mknod creates a FIFO (named pipe), character special file, or block special file with the specified *name*.

A special file is a triple (boolean, integer, integer) stored in the filesystem. The boolean chooses between character special file and block special file. The two integers are the major and minor device number.

Thus, a special file takes almost no place on disk, and is used only for communication with the operating system, not for data storage. Often special files refer to hardware devices (disk, tape, tty, printer) or to operating system services (/dev/null, /dev/random).

Block special files usually are disk-like devices (where data can be accessed given a block number, and e.g. it is meaningful to have a block cache). All other devices are character special devices. (Long ago the distinction was a different one: I/O to a character special file would be unbuffered, to a block special file buffered.)

The **mknod** command is what creates files of this type.

The argument following *name* specifies the type of file to make:

p for a FIFO

b for a block (buffered) special file

c for a character (unbuffered) special file

The GNU version of **mknod** allows **u** (`unbuffered') as a synonym for **c**.

When making a block or character special file, the major and minor device numbers must be given after the file type (in decimal, or in octal with leading 0; the GNU version also allows hexadecimal with leading 0x). By default, the mode of created files is 0666 (`a+rw') minus the bits set in the umask.

OPTIONS

-**m** *mode*, --**mode=***mode*
Set the mode of created directories to *mode*, which can be symbolic as in **chmod**(1) and then uses the default mode as the point of departure.

GNU STANDARD OPTIONS

--**help**
Print a usage message on standard output and exit successfully.

--**version**
Print version information on standard output, then exit successfully.

-- Terminate option list.

CONFORMING TO

POSIX does not describe this command as it is nonportable, and recommends using **mkfifo**(1) to make FIFOs. SVID has a command */etc/mknod* with the above syntax, but without the mode option.

NOTES

On a Linux system (version 1.3.22 or newer) the file */usr/src/linux/Documentation/devices.tex* contains a list of devices with device name, type, major and minor number.

The present page describes **mknod** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.

SEE ALSO

chmod(1), **mkfifo**(1), **mknod**(2)


TOC

NAME

mv - move (rename) files

SYNOPSIS

mv [*option*...] *source target*
mv [*option*...] *source*... *target*

POSIX options: **[-fi]**

GNU options (shortest form): **[-bfiuv] [-S** *suffix***] [-V {numbered,existing,simple}] [--help] [--version] [--]**

DESCRIPTION

mv moves or renames files or directories.

If the last argument names an existing directory, **mv** moves each other given file into a file with the same name in that directory. Otherwise, if only two files are given, it renames the first as the second. It is an error if the last argument is not a directory and more than two files are given.

Thus, `mv /a/x/y /b' will rename the file /a/x/y into /b/y if /b was an existing directory, and into /b otherwise.

Let us call the file a given file is going to be moved into its *destination*. If *destination* exists, and either the -i option is given, or *destination* is unwritable, standard input is a terminal, and the **-f** option is not given, **mv** prompts the user for whether to replace the file, writing a question to stderr and reading an answer from stdin. If the

response is not affirmative, the file is skipped.

When both *source* and *destination* are on the same filesystem, they are the same file (just the name is changed; owner, mode, timestamps remain unchanged). When they are on different filesystems, the source file is copied and then deleted. **mv** will copy modification time, access time, user and group ID, and mode if possible. When copying user and/or group ID fails, the setuid and setgid bits are cleared in the copy.

POSIX OPTIONS

-f Do not prompt for confirmation.

-i Prompt for confirmation when *destination* exists. (In case both -f and -i are given, the last one given takes effect.)

GNU DETAILS

The GNU implementation (in fileutils-3.16) is broken in the sense that **mv** can move only regular files across filesystems.

GNU OPTIONS

-f, --force
Remove existing destination files and never prompt the user.

-i, --interactive
Prompt whether to overwrite existing regular destination files. If the response does not begin with `y' or `Y', the file is skipped.

-u, --update
Do not move a nondirectory that has an existing destination with the same or newer modification time.

-v, --verbose
Print the name of each file before moving it.

GNU BACKUP OPTIONS

The GNU versions of programs like **cp**, **mv**, **ln**, **install** and **patch** will make a backup of files about to be overwritten, changed or destroyed if that is desired. That backup files are desired is indicated by the -b option. How they should be named is specified by the -V option. In case the name of the backup file is given by the name of the file extended by a suffix, this suffix is specified by the -S option.

-b, --backup
Make backups of files that are about to be overwritten or removed.

-S *SUFFIX*, **--suffix=***SUFFIX*
Append *SUFFIX* to each backup file made. If this option is not specified, the value of the **SIMPLE_BACKUP_SUFFIX** environment variable is used. And if **SIMPLE_BACKUP_SUFFIX** is not set, the default is `~'.

-V *METHOD*, **--version-control=***METHOD*
Specify how backup files are named. The *METHOD* argument can be `numbered' (or `t'), `existing' (or `nil'), or `never' (or `simple'). If this option is not specified, the value of the **VERSION_CONTROL** environment variable is used. And if **VERSION_CONTROL** is not set, the default backup type is `existing'.

This option corresponds to the Emacs variable `version-control'. The valid *METHOD*s are (unique abbreviations are accepted):

t, **numbered**
Always make numbered backups.

nil, **existing**
Make numbered backups of files that already have them, simple backups of the others.

never, **simple**
Always make simple backups.

GNU STANDARD OPTIONS

--**help**
Print a usage message on standard output and exit successfully.

--**version**
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_COLLATE, LC_CTYPE and LC_MESSAGES have the usual meaning. For the GNU version, the variables SIMPLE_BACKUP_SUFFIX and VERSION_CONTROL control backup file naming, as described above.

CONFORMING TO

POSIX 1003.2, except that directory hierarchies cannot be moved across filesystems.

NOTES

This page describes **mv** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.

NAME

rm - remove files or directories

SYNOPSIS

rm [*options***]** *file*...

POSIX options: **[-fiRr]**

GNU options (shortest form): **[-dfirvR] [--help] [--version] [--]**

DESCRIPTION

rm removes each given *file*. By default, it does not remove directories. But when the -r or -R option is given, the entire directory tree below the specified directory is removed (and there are no limitations on the depth of directory trees that can be removed by `rm -r'). It is an error when the last path component of *file* is either . or .. (so as to avoid unpleasant surprises with `rm -r .*' or so).

If the -i option is given, or if a file is unwritable, standard input is a terminal, and the -**f** option is not given, **rm** prompts the user for whether to remove the file, writing a question to stderr and reading an answer from stdin. If the response is not affirmative, the file is skipped.

POSIX OPTIONS

-f Do not prompt for confirmation. Do not write diagnostic messages. Do not produce an error return status if the only errors were nonexisting files.

-i Prompt for confirmation. (In case both -f and -i are given, the last one given takes effect.)

-**r** or -**R**
Recursively remove directory trees.

SVID DETAILS

The System V Interface Definition forbids removal of the last link to an executable binary file that is being executed.

GNU DETAILS

The GNU implementation (in fileutils-3.16) is broken in the sense that there is an upper limit to the depth of hierarchies that can be removed. (If necessary, a utility `deltree' can be used to remove very deep trees.)

GNU OPTIONS

-d, --directory
Remove directories with **unlink**(2) instead of **rmdir**(2), and don't require a directory to be empty before trying to unlink it. Only works if you have appropriate privileges. Because unlinking a directory causes any files in the deleted directory to become unreferenced,

it is wise to **fsck**(8) the filesystem after doing this.

-f, --force
Ignore nonexistent files and never prompt the user.

-i, --interactive
Prompt whether to remove each file. If the response does not begin with `y' or `Y', the file is skipped.

-r, -R, --recursive
Remove the contents of directories recursively.

-v, --verbose
Print the name of each file before removing it.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_COLLATE, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2, except for the limitation on file hierarchy depth.

NOTES

This page describes **rm** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.

NAME

rmdir - remove empty directories

SYNOPSIS

rmdir [*options*] *directory*...

POSIX options: **[-p]**

GNU options (shortest form): **[-p] [--help] [--version] [--]**

DESCRIPTION

rmdir removes empty directories.

If any *directory* argument does not refer to an existing empty directory, it is an error.

POSIX OPTIONS

-p If *directory* includes more than one pathname component, remove it, then strip the last component and remove the resulting directory, etc., until all components have been removed. Thus, `rmdir -p a/b/c' is equivalent to `rmdir a/b/c; rmdir a/b; rmdir a'.

GNU OPTIONS

-p, --parents
As above.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2

EXAMPLE OF USE

The command `*rmdir foo*' will remove the directory *foo* if it

is empty. To remove a nonempty directory, together with everything below, use `*rm* -**r** *foo*'.

NOTES

This page describes **rmdir** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.


TOC

NAME

touch - change file timestamps

SYNOPSIS

touch [-acm][-r *ref_file*|**-t** *time*] *file*...

Obsolescent version:
touch [-acm][*ugly_time***]** *file*...

GNU version:
touch [-acfm] [-r *file***] [-t** *decimtime***] [-d** *time***] [--time={atime,access,use,mtime,modify}] [--date=***time***] [--reference=***file***] [--no-create] [--help] [--version] [--]** *file*...

DESCRIPTION

touch changes the access and/or modification timestamps of each specified *file*. These timestamps are changed to the current time, unless the -r option is specified, in which case they are changed to the corresponding timestamps of the file *ref_file*, or the -t option is specified, in which case they are changed to the specified *time*. Both times are changed when neither or both of the -a and -m options are given. Only the access or only the modification time is changed when one of the options -a and -m is given. If the file did not exist yet, it is created (as an empty file with mode 0666, modified by the umask), unless the -c option is given.

POSIX OPTIONS

-a Change the access time of *file*.

-c Do not create *file*.

-m Change the modification time of *file*.

-r *ref_file*
Use the corresponding timestamp of *ref_file* as the new value for the changed timestamp(s).

-t *time*
Use the specified time as the new value for the changed timestamp(s). The argument is a decimal number of the form
[[CC]YY]MMDDhhmm[.SS]
with the obvious meaning. If CC is not specified, the year CCYY is taken to be in the range 1969-2068. If SS is not specified, it is taken to be 0. It may be specified in the range 0-61 so that it is possible to refer to leap seconds. The resulting time is taken as a time for the time zone specified by the environment variable TZ. It is an error if the resulting time precedes 1 January 1970.

POSIX DETAILS

The second form of invocation has the disadvantage that there is some ambiguity as to whether *ugly_time* is a time or a file argument. It is taken to be a time when no -r or -t option is present, there are at least two arguments, and the first argument is an eight- or ten-digit decimal integer. The format of *ugly_time* is MMDDhhmm[yy], where an yy in the range 69-99 denotes a year in the range 1969-1999, and an unspecified yy denotes the current year. This form is obsolete.

GNU DETAILS

If the first *file* would be a valid argument to the **-t** option and no timestamp is given with any of the **-d**, **-r** or **-t** options and the `--' argument is not given, that argument is interpreted as the time for the other files instead of as a file name.

If changing both the access and modification times to the current time, **touch** can change the timestamps for files that the user running it does not own but has write permission for. Otherwise, the user must own the files.

GNU OPTIONS

-a, --time=atime, --time=access, --time=use
Change the access time only.

-c, --no-create
Do not create files that do not exist.

-d, --date=*time*
Use *time* instead of the current time. It can contain month names, timezones, `am' and `pm', etc.

-f Ignored; for compatibility with BSD versions of **touch**(1).

-m, --time=mtime, --time=modify
Change the modification time only.

-r *file*, **--reference=***file*
Use the times of the reference *file* instead of the current time.

-t *decimtime*
Here *decimtime* has the format MMDDhhmm[[CC]YY][.ss] Use the argument (months, days, hours, minutes, optional century and years, optional seconds) instead of the current time. Note that this format violates the POSIX

specification.

GNU STANDARD OPTIONS

--help
Print a usage message on standard output and exit successfully.

--version
Print version information on standard output, then exit successfully.

-- Terminate option list.

ENVIRONMENT

The variable TZ is used to interpret explicitly given times. The variables LANG, LC_ALL, LC_CTYPE and LC_MESSAGES have the usual meaning.

CONFORMING TO

POSIX 1003.2 describes a syntax for the argument of the -t option that differs from that used by the GNU implementation.

EXAMPLE OF USE

The command `*touch foo*' will create the file *foo* if it didn't exist, and change the time of last modification to now. It is often used to guide the actions of **make**.

NOTES

This page describes **touch** as found in the fileutils-3.16 package; other versions may differ slightly. Mail corrections and additions to aeb@cwi.nl and aw@mail1.bet1.puv.fi and ragnar@lightside.ddns.org . Report bugs in the program to fileutils-bugs@gnu.ai.mit.edu.

Linux Man Pages Section 2

- _exit.2
- _llseek.2
- _newselect.2
- _sysctl.2
- accept.2
- access.2
- acct.2
- adjtimex.2
- afs_syscall.2
- alarm.2
- bdflush.2
- bind.2
- break.2
- brk.2
- cacheflush.2
- chdir.2
- chmod.2
- chown.2
- chroot.2
- clone.2
- close.2
- connect.2
- creat.2
- dup.2
- dup2.2
- execve.2
- exit.2
- fchdir.2
- fchmod.2
- fchown.2
- fcntl.2

- fdatasync.2
- flock.2
- fork.2
- fstat.2
- fstatfs.2
- fsync.2
- ftruncate.2
- getdents.2
- getdomainname.2
- getdtablesize.2
- getegid.2
- geteuid.2
- getgid.2
- getgroups.2
- gethostid.2
- gethostname.2
- getitimer.2
- getpagesize.2
- getpeername.2
- getpgid.2
- getpgrp.2
- getpid.2
- getppid.2
- getpriority.2
- getresgid.2
- getresuid.2
- getrlimit.2
- getrusage.2
- getsid.2
- getsockname.2
- getsockopt.2
- gettimeofday.2
- getuid.2

- gtty.2
- idle.2
- intro.2
- ioctl.2
- ioctl_list.2
- ioperm.2
- iopl.2
- ipc.2
- kill.2
- killpg.2
- lchown.2
- link.2
- listen.2
- llseek.2
- lock.2
- lseek.2
- lstat.2
- mkdir.2
- mknod.2
- mlock.2
- mlockall.2
- mmap.2
- modify_ldt.2
- mount.2
- mprotect.2
- mpx.2
- mremap.2
- msgctl.2
- msgget.2
- msgop.2
- msgrcv.2
- msgsnd.2
- msync.2

- munlock.2
- munlockall.2
- munmap.2
- nanosleep.2
- nfsservctl.2
- nice.2
- obsolete.2
- oldfstat.2
- oldlstat.2
- oldolduname.2
- oldstat.2
- olduname.2
- open.2
- outb.2
- pause.2
- personality.2
- pipe.2
- poll.2
- prctl.2
- prof.2
- ptrace.2
- quotactl.2
- read.2
- readdir.2
- readlink.2
- readv.2
- reboot.2
- recv.2
- recvfrom.2
- recvmsg.2
- rename.2
- rmdir.2
- sbrk.2

- sched_get_priority_max.2
- sched_get_priority_min.2
- sched_getparam.2
- sched_getscheduler.2
- sched_rr_get_interval.2
- sched_setparam.2
- sched_setscheduler.2
- sched_yield.2
- select.2
- semctl.2
- semget.2
- semop.2
- send.2
- sendmsg.2
- sendto.2
- setdomainname.2
- setegid.2
- seteuid.2
- setfsgid.2
- setfsuid.2
- setgid.2
- setgroups.2
- sethostid.2
- sethostname.2
- setitimer.2
- setpgid.2
- setpgrp.2
- setpriority.2
- setregid.2
- setresgid.2
- setresuid.2
- setreuid.2
- setrlimit.2

- setsid.2
- setsockopt.2
- settimeofday.2
- setuid.2
- setup.2
- sgetmask.2
- shmat.2
- shmctl.2
- shmdt.2
- shmget.2
- shmop.2
- shutdown.2
- sigaction.2
- sigblock.2
- siggetmask.2
- sigmask.2
- signal.2
- sigpause.2
- sigpending.2
- sigprocmask.2
- sigreturn.2
- sigsetmask.2
- sigsuspend.2
- sigvec.2
- socket.2
- socketcall.2
- socketpair.2
- ssetmask.2
- stat.2
- statfs.2
- stime.2
- stty.2
- swapoff.2

- swapon.2
- symlink.2
- sync.2
- syscalls.2
- sysctl.2
- sysfs.2
- sysinfo.2
- syslog.2
- time.2
- times.2
- truncate.2
- umask.2
- umount.2
- uname.2
- undocumented.2
- unimplemented.2
- unlink.2
- uselib.2
- ustat.2
- utime.2
- utimes.2
- vfork.2
- vhangup.2
- vm86.2
- wait.2
- wait3.2
- wait4.2
- waitpid.2
- write.2
- writev.2

NAME

_exit - terminate the current process

SYNOPSIS

#include <unistd.h>

void _exit(int *status)*);

DESCRIPTION

_exit terminates the calling process immediately. Any open file descriptors belonging to the process are closed; any children of the process are inherited by process 1, init, and the process's parent is sent a **SIGCHLD** signal.

status is returned to the parent process as the process's exit status, and can be collected using one of the **wait** family of calls.

RETURN VALUE

_exit never returns.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, BSD 4.3

NOTES

_exit does not call any functions registered with the ANSI C **atexit** function and does not flush standard I/O buffers. To do these things, use **exit**(3).

SEE ALSO

fork(2), **execve**(2), **waitpid**(2), **wait**(2), **exit**(3)

NAME

accept - accept a connection on a socket

SYNOPSIS

#include <sys/types.h>

#include <sys/socket.h>

int accept(int s, struct sockaddr ***addr*****, int** ***addrlen***

DESCRIPTION

The argument **s** is a socket that has been created with **socket**(2), bound to an address with **bind**(2), and is listening for connections after a **listen**(2). The **accept** function extracts the first connection request on the queue of pending connections, creates a new socket with the same properties of **s** and allocates a new file descriptor for the socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, **accept** blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, **accept** returns an error as described below. The accepted socket may not be used to accept more connections. The original socket **s** remains open.

The argument *addr* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the *addr* parameter is determined by the domain in which the communication is occurring. The *addrlen* is a value-result parameter; it should initially contain the amount of space pointed to by

addr; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types, currently with **SOCK_STREAM**.

It is possible to **select**(2) a socket for the purposes of doing an **accept** by selecting it for read.

For certain protocols which require an explicit confirmation, such as **ISO** or **DATAKIT**, **accept** can be thought of as merely dequeuing the next connection request and not implying confirmation. Confirmation can be implied by a normal read or write on the new file descriptor, and rejection can be implied by closing the new socket.

One can obtain user connection request data without confirming the connection by issuing a **recvmsg**(2) call with an *msg_iovlen* of 0 and a non-zero *msg_controllen*, or by issuing a **getsockopt**(2) request. Similarly, one can provide user connection rejection information by issuing a **sendmsg**(2) call with providing only the control information, or by calling **setsockopt**(2).

RETURN VALUES

The call returns -1 on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.

ERRORS

The BSD man page documents five possible error returns.

EBADF
The descriptor is invalid.

ENOTSOCK
The descriptor references a file, not a socket.

EOPNOTSUPP
The referenced socket is not of type **SOCK_STREAM**.

EFAULT
The *addr* parameter is not in a writable part of the user address space.

EWOULDBLOCK
The socket is marked non-blocking and no connections are present to be accepted.

Various Linux kernels can return various other errors such as **EMFILE**, **EINVAL**, **ENOSR**, **ENOBUFS**, **EAGAIN**, **EPERM**, **ECONNABORTED**, **ESOCKTNOSUPPORT**, **EPROTONOSUPPORT**, **ETIMEDOUT**, **ERESTARTSYS**.

CONFORMING TO

SVr4, 4.4BSD (the **accept** function first appeared in BSD 4.2). IRIX documents additional errors EMFILE and ENFILE. Solaris documents additional errors EINTR, ENODEV, ENOMEM, ENOSR and EPROTO.

SEE ALSO

bind(2), **connect**(2), **listen**(2),


TOC

NAME

access - check user's permissions for a file

SYNOPSIS

```
#include <unistd.h>

int access(const char *pathname, int mode));
```

DESCRIPTION

access checks whether the process would be allowed to read, write or test for existence of the file (or other file system object) whose name is *pathname*. If *pathname* is a symbolic link permissions of the file referred to by this symbolic link are tested.

mode is a mask consisting of one or more of **R_OK**, **W_OK**, **X_OK** and

R_OK, **W_OK** and **X_OK** request checking whether the file exists and has read, write and execute permissions, respectively. **F_OK** just requests checking for the existence of the file.

The tests depend on the permissions of the directories occurring in the path to the file, as given in *pathname*, and on the permissions of directories and files referred to by symbolic links encountered on the way.

The check is done with the process's *real* uid and gid, rather than with the effective ids as is done when actually attempting an operation. This is to allow set-UID programs to easily determine the invoking user's authority.

Only access bits are checked, not the file type or contents. Therefore, if a directory is found to be "writable," it probably means that files can be created in the directory, and not that the directory can be written as a file. Similarly, a DOS file may be found to be "executable," but the **execve**(2) call will still fail.

RETURN VALUE

On success (all requested permissions granted), zero is returned. On error (at least one bit in *mode* asked for a permission that is denied, or some other error occurred), -1 is returned, and *errno* is set appropriately.

ERRORS

EACCES The requested access would be denied to the file or search permission is denied to one of the directories in *pathname*.

EROFS Write permission was requested for a file on a read-only filesystem.

EFAULT *pathname* points outside your accessible address space.

EINVAL *mode* was incorrectly specified.

ENAMETOOLONG
pathname is too long.

ENOENT A directory component in *pathname* would have been accessible but does not exist or was a dangling symbolic link.

ENOTDIR A component used as a directory in *pathname* is not, in fact, a directory.

ENOMEM Insufficient kernel memory was available.

ELOOP Too many symbolic links were encountered in resolving *pathname*.

EIO An I/O error occurred.

RESTRICTIONS

access returns an error if any of the access types in the requested call fails, even if other types might be successful.

access may not work correctly on NFS file systems with UID mapping enabled, because UID mapping is done on the server and hidden from the client, which checks permissions.

Using **access** to check if a user is authorized to e.g. open a file before actually doing so using **open**(2) creates a security hole, because the user might exploit the short time interval between checking and opening the file to manipulate it.

CONFORMING TO

SVID, AT&T, POSIX, X/OPEN, BSD 4.3

SEE ALSO

stat(2), **open**(2), **chmod**(2), **setuid**(2), **setgid**(2).


TOC

NAME

acct - switch process accounting on or off

SYNOPSIS

```
#include <unistd.h>

int acct(const char *filename));
```

DESCRIPTION

When called with the name of an existing file as argument, accounting is turned on, records for each terminating process are appended to *filename* as it terminates. An argument of **NULL** causes accounting to be turned off.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

ENOSYS BSD process accounting has not been enabled when the operating system kernel was compiled. The kernel

configuration parameter controlling this feature is CONFIG_BSD_PROCESS_ACCT.

ENOMEM Out of memory.

EPERM The calling process has no permission to enable process accounting.

EACCESS The argument *filename* is not a regular file.

EIO Error writing to the file *filename*.

EUSERS There are no more free file structures or we run out of memory.

CONFORMING TO

SVr4 (but not POSIX). SVr4 documents EACCES, EBUSY, EFAULT, ELOOP, ENAMETOOLONG, ENOTDIR, ENOENT, EPERM and EROFS error conditions, but no ENOSYS.

NOTES

No accounting is produced for programs running when a crash occurs. In particular, nonterminating processes are never accounted for.

NAME

adjtimex - tune kernel clock

SYNOPSIS

```
#include <sys/timex.h>

int adjtimex(struct timex *buf));
```

DESCRIPTION

Linux uses David L. Mills' clock adjustment algorithm. **adjtimex** reads and optionally sets adjustment parameters for this algorithm.

adjtimex takes a pointer to a *timex* structure, updates kernel parameters from field values, and returns the same structure with current kernel values. This structure is declared as follows:

```
struct timex
{
    int modes;           /* mode selector */
    long offset;         /* time offset (usec) */
    long freq;           /* frequency offset (scaled ppm) */
    long maxerror;       /* maximum error (usec) */
    long esterror;       /* estimated error (usec) */
    int status;          /* clock command/status */
    long constant;       /* pll time constant */
    long precision;      /* clock precision (usec) (read only) */
    long tolerance;      /* clock frequency tolerance (ppm)
                            (read only) */
    struct timeval time; /* current time (read only) */
    long tick;           /* usecs between clock ticks */
};
```

The *modes* field determines which parameters, if any, to set. It may contain a bitwise-*or* combination of zero or more of the following bits:

```
#define ADJ_OFFSET            0x0001 /* time offset */
#define ADJ_FREQUENCY         0x0002 /* frequency offset */
#define ADJ_MAXERROR          0x0004 /* maximum time error */
#define ADJ_ESTERROR          0x0008 /* estimated time error */
#define ADJ_STATUS            0x0010 /* clock status */
#define ADJ_TIMECONST         0x0020 /* pll time constant */
#define ADJ_TICK              0x4000 /* tick value */
#define ADJ_OFFSET_SINGLESHOT 0x8001 /* old-fashioned adjtime */
```

Ordinary users are restricted to a zero value for *mode*. Only the superuser may set any parameters.

RETURN VALUE

On success, **adjtimex** returns the clock state:

```
#define TIME_OK   0 /* clock synchronized */
#define TIME_INS  1 /* insert leap second */
#define TIME_DEL  2 /* delete leap second */
#define TIME_OOP  3 /* leap second in progress */
#define TIME_WAIT 4 /* leap second has occurred */
#define TIME_BAD  5 /* clock not synchronized */
```

On failure, **adjtimex** returns -1 and sets *errno*.

ERRORS

EFAULT
buf does not point to writeable memory.

EPERM
buf.mode is non-zero and the user is not super-user.

EINVAL
An attempt is made to set *buf.offset* to a value outside

the range -131071 to +131071, or to set *buf.status* to a value other than those listed above, or to set *buf.tick* to a value outside the range 900000/**HZ** to 1100000/**HZ**, where **HZ** is the system timer interrupt frequency.

CONFORMING TO

adjtimex is Linux specific and should not be used in programs intended to be portable. There is a similar but less general call **adjtime** in SVr4.

SEE ALSO

settimeofday(2).

NAME

alarm - set an alarm clock for delivery of a signal

SYNOPSIS

#include <unistd.h>

unsigned int alarm(unsigned int *seconds)*);

DESCRIPTION

alarm arranges for a **SIGALRM** signal to be delivered to the process in *seconds* seconds.

If *seconds* is zero, no new **alarm** is scheduled.

In any event any previously set **alarm** is cancelled.

RETURN VALUE

alarm returns the number of seconds remaining until any previously scheduled alarm was due to be delivered, or zero if there was no previously scheduled alarm.

NOTES

alarm and **setitimer** share the same timer; calls to one will interfere with use of the other.

Scheduling delays can, as ever, cause the execution of the process to be delayed by an arbitrary amount of time.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, BSD 4.3

SEE ALSO

setitimer(2), **signal**(2), **sigaction**(2), **gettimeofday**(2), **select**(2), **pause**(2),

NAME

bdflush - start, flush, or tune buffer-dirty-flush daemon

SYNOPSIS

int bdflush(int *func*, **long** ***address)**;
int bdflush(int *func*, **long** *data)*;

DESCRIPTION

bdflush starts, flushes, or tunes the buffer-dirty-flush daemon. Only the super-user may call **bdflush**.

If *func* is negative or 0, and no daemon has been started, then **bdflush** enters the daemon code and never returns.

If *func* is 1, some dirty buffers are written to disk.

If *func* is 2 or more and is even (low bit is 0), then *address* is the address of a long word, and the tuning parameter numbered (*func*-2)/2 is returned to the caller in that address.

If *func* is 3 or more and is odd (low bit is 1), then *data* is a long word, and the kernel sets tuning parameter numbered (*func*-3)/2 to that value.

The set of parameters, their values, and their legal ranges are defined in the kernel source file *fs/buffer.c*.

RETURN VALUE

If *func* is negative or 0 and the daemon successfully starts, **bdflush** never returns. Otherwise, the return value is 0 on success and -1 on failure, with *errno* set to indicate the error.

ERRORS

EPERM Caller is not super-user.

EFAULT *address* points outside your accessible address space.

EBUSY An attempt was made to enter the daemon code after another process has already entered.

EINVAL An attempt was made to read or write an invalid parameter number, or to write an invalid value to a parameter.

CONFORMING TO

bdflush is Linux specific and should not be used in programs intended to be portable.

SEE ALSO

fsync(2), **sync**(2), **update**(8), **sync**(8).


TOC


TOC

NAME

bind - bind a name to a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, int addrlen
```

DESCRIPTION

bind gives the socket, *sockfd*, the local address *my_addr*. *my_addr* is *addrlen* bytes long. Traditionally, this is called "assigning a name to a socket" (when a socket is created with **socket**(2), it exists in a name space (address family) but has no name assigned.)

NOTES

Binding a name in the UNIX domain creates a socket in the file system that must be deleted by the caller when it is no longer needed (using **unlink**(2)).

The rules used in name binding vary between communication domains. Consult the manual entries in section 4 for detailed information.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF *sockfd* is not a valid descriptor.

EINVAL The socket is already bound to an address. This may change in the future: see *linux/unix/sock.c* for details.

EACCES The address is protected, and the user is not the super-user.

ENOTSOCK
Argument is a descriptor for a file, not a socket.

The following errors are specific to UNIX domain (AF_UNIX) sockets:

EINVAL The *addr_len* was wrong, or the socket was not in the **AF_UNIX** family.

EROFS The socket inode would reside on a read-only file system.

EFAULT *my_addr* points outside your accessible address space.

ENAMETOOLONG
my_addr is too long.

ENOENT The file does not exist.

ENOMEM Insufficient kernel memory was available.

ENOTDIR A component of the path prefix is not a directory.

EACCES Search permission is denied on a component of the path prefix.

ELOOP Too many symbolic links were encountered in resolving *my_addr*.

CONFORMING TO

SVr4, 4.4BSD (the **bind** function first appeared in BSD 4.2). SVr4 documents additional EADDRNOTAVAIL, EADDRINUSE, ENOSR general error conditions, and additional EIO, EISDIR and EROFS Unix-domain error conditions.

SEE ALSO

accept(2), **connect**(2), **listen**(2), **socket**(2), **getsockname**(2)


TOC

NAME

brk, sbrk - change data segment size

SYNOPSIS

```
#include <unistd.h>

int brk(void *end_data_segment));

void *sbrk(ptrdiff_t increment));
```

DESCRIPTION

brk sets the end of the data segment to the value specified by *end_data_segment*. *end_datasegment* must be greater than end of the text segment and it must be 16kB before the end of the stack.

sbrk increments the program's data space by *increment* bytes. **sbrk** isn't a system call, it is just a C library wrapper.

RETURN VALUE

On success, **brk** returns zero, and **sbrk** returns a pointer to the start of the new area. On error, -1 is returned, and *errno* is set to **ENOMEM**.

CONFORMING TO

BSD 4.3

brk and **sbrk** are not defined in the C Standard and are deliberately excluded from the POSIX.1 standard (see paragraphs B.1.1.1.3 and B.8.3.3).

SEE ALSO

execve(2), **getrlimit**(2), **malloc**(3)

NAME

cacheflush - flush contents of instruction and/or data cache

SYNOPSIS

#include <asm/cachectl.h>

int cacheflush(char *_addr_, int _nbytes_, int _cache_

DESCRIPTION

cacheflush flushes contents of indicated cache(s) for user addresses in the range addr to (addr+nbytes-1). Cache may be one of:

ICACHE
Flush the instruction cache.

DCACHE
Write back to memory and invalidate the affected valid cache lines.

BCACHE
Same as **(ICACHE|DCACHE).**

RETURN VALUE

cacheflush returns 0 on success or -1 on error. If errors are detected, errno will indicate the error.

ERRORS

EINVAL
cache parameter is not one of ICACHE, DCACHE, or BCACHE.

EFAULT
Some or all of the address range addr to (addr+nbytes-1) is not accessible.

BUGS

The current implementation ignores the addr and nbytes parameters. Therefore always the whole cache is flushed.

NOTE

This system call is only available on MIPS based systems. It should not be used in programs intended to be portable.

NAME

chdir, fchdir - change working directory

SYNOPSIS

```
#include <unistd.h>

int chdir(const char *path));
int fchdir(int fd));
```

DESCRIPTION

chdir changes the current directory to that specified in *path*.

fchdir is identical to **chdir**, only that the directory is given as an open file descriptor.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

Depending on the file system, other errors can be returned. The more general errors for **chdir** are listed below:

EFAULT *path* points outside your accessible address space.

ENAMETOOLONG
path is too long.

ENOENT The file does not exist.

ENOMEM Insufficient kernel memory was available.

ENOTDIR A component of *path* is not a directory.

EACCES Search permission is denied on a component of *path*.

ELOOP Too many symbolic links were encountered in resolving *path*.

EIO An I/O error occurred.

The general errors for **fchdir** are listed below:

EBADF *fd* is not a valid file descriptor. **EACCES** Search permission was denied on the directory open on *fd*.

CONFORMING TO

The **chdir** call is compatible with SVr4, SVID, POSIX, X/OPEN, 4.4BSD. SVr4 documents additional EINTR, ENOLINK, and EMULTIHOP error conditions but has no ENOMEM. POSIX.1 does not have ENOMEM or ELOOP error conditions. X/OPEN does not have EFAULT, ENOMEM or EIO error conditions.

The **fchdir** call is compatible with SVr4, 4.4BSD and X/OPEN. SVr4 documents additional EIO, EINTR, and ENOLINK error conditions. X/OPEN documents additional EINTR and EIO error conditions.

SEE ALSO

getcwd(3), **chroot**(2)


TOC

NAME

chmod, fchmod - change permissions of a file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod(const char *path, mode_t mode));
int fchmod(int fildes, mode_t mode));
```

DESCRIPTION

The mode of the file given by *path* or referenced by *fildes* is changed.

Modes are specified by *or'ing* the following:

```
     S_ISUID   04000 set user ID on execution

     S_ISGID   02000 set group ID on execution

     S_ISVTX   01000 sticky bit

     S_IRUSR (S_IREAD)
               00400 read by owner

     S_IWUSR (S_IWRITE)
               00200 write by owner

     S_IXUSR (S_IEXEC)
               00100 execute/search by owner
```

S_IRGRP 00040 read by group

S_IWGRP 00020 write by group

S_IXGRP 00010 execute/search by group

S_IROTH 00004 read by others

S_IWOTH 00002 write by others

S_IXOTH 00001 execute/search by others

The effective UID of the process must be zero or must match the owner of the file.

If the effective UID of the process is not zero and the group of the file does not match the effective group ID of the process or one of its supplementary group IDs, the S_ISGID bit will be turned off, but this will not cause an error to be returned.

Depending on the file system, set user ID and set group ID execution bits may be turned off if a file is written. On some file systems, only the super-user can set the sticky bit, which may have a special meaning (e.g., for directories, a file can only be deleted by the owner or the super-user).

On NFS file systems, restricting the permissions will immediately influence already open files, because the access control is done on the server, but open files are maintained by the client. Widening the permissions may be delayed for other clients if attribute caching is enabled on them.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

Depending on the file system, other errors can be returned. The more general errors for **chmod** are listed below:

EPERM The effective UID does not match the owner of the file, and is not zero.

EROFS The named file resides on a read-only file system.

EFAULT *path* points outside your accessible address space.

ENAMETOOLONG
path is too long.

ENOENT The file does not exist.

ENOMEM Insufficient kernel memory was available.

ENOTDIR A component of the path prefix is not a directory.

EACCES Search permission is denied on a component of the path prefix.

ELOOP Too many symbolic links were encountered in resolving *path*.

EIO An I/O error occurred.

The general errors for **fchmod** are listed below:

EBADF The file descriptor *fildes* is not valid.

EROFS See above.

EPERM See above.

EIO See above.

CONFORMING TO

The **chmod** call conforms to SVr4, SVID, POSIX, X/OPEN, 4.4BSD. SVr4 documents EINTR, ENOLINK and EMULTIHOP returns, but no ENOMEM. POSIX.1 does not document EFAULT, ENOMEM, ELOOP or EIO error conditions, or the macros **S_IREAD**, **S_IWRITE** and **S_IEXEC**.

The **fchmod** call conforms to 4.4BSD and SVr4. SVr4 documents additional EINTR and ENOLINK error conditions. POSIX requires the **fchmod** function if at least one of **_POSIX_MAPPED_FILES** and **_POSIX_SHARED_MEMORY_OBJECTS** is defined, and documents additional ENOSYS and EINVAL error conditions, but does not document EIO.

POSIX and X/OPEN do not document the sticky bit.

SEE ALSO

open(2), **chown**(2), **stat**(2)


TOC

NAME

chown, fchown, lchown - change ownership of a file

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

int chown(const char *path, uid_t owner, gid_t group
int fchown(int fd, uid_t owner, gid_t group
int lchown(const char *path, uid_t owner, gid_t group
```

DESCRIPTION

The owner of the file specified by *path* or by *fd* is changed. Only the super-user may change the owner of a file. The owner of a file may change the group of the file to any group of which that owner is a member. The super-user may change the group arbitrarily.

If the *owner* or *group* is specified as -1, then that ID is not changed.

When the owner or group of an executable file are changed by a non-super-user, the S_ISUID and S_ISGID mode bits are cleared. POSIX does not specify whether this also should happen when root does the *chown*; the Linux behaviour depends on the kernel version. In case of a non-group-executable file (with clear S_IXGRP bit) the S_ISGID bit indicates mandatory locking, and is not cleared by a *chown*.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

Depending on the file system, other errors can be returned. The more general errors for **chown** are listed below:

EPERM The effective UID does not match the owner of the file, and is not zero; or the *owner* or *group* were specified incorrectly.

EROFS The named file resides on a read-only file system.

EFAULT *path* points outside your accessible address space.

ENAMETOOLONG
path is too long.

ENOENT The file does not exist.

ENOMEM Insufficient kernel memory was available.

ENOTDIR A component of the path prefix is not a directory.

EACCES Search permission is denied on a component of the path prefix.

ELOOP Too many symbolic links were encountered in resolving *path*.

The general errors for **fchown** are listed below:

EBADF The descriptor is not valid.

ENOENT See above.

EPERM See above.

EROFS See above.

EIO A low-level I/O error occurred while modifying the inode.

NOTES

In versions of Linux prior to 2.1.81 (and distinct from 2.1.46), **chown** did not follow symbolic links. Since Linux 2.1.81, **chown** does follow symbolic links, and there is a new system call **lchown** that does not follow symbolic links. Since Linux 2.1.86, this new call (that has the same semantics as the old **chown**) has got the same syscall number, and **chown** got the newly introduced number.

The prototype for **fchown** is only available if **__USE_BSD** is defined.

CONFORMING TO

The **chown** call conforms to SVr4, SVID, POSIX, X/OPEN. The 4.4BSD version can only be used by the superuser (that is, ordinary users cannot give away files). SVr4 documents EINVAL, EINTR, ENOLINK and EMULTIHOP returns, but no ENOMEM. POSIX.1 does not document ENOMEM or ELOOP error conditions.

The **fchown** call conforms to 4.4BSD and SVr4. SVr4 documents additional EINVAL, EIO, EINTR, and ENOLINK error conditions.

RESTRICTIONS

The **chown**() semantics are deliberately violated on NFS file systems which have UID mapping enabled. Additionally, the

semantics of all system calls which access the file contents are violated, because **chown**() may cause immediate access revocation on already open files. Client side caching may lead to a delay between the time where ownership have been changed to allow access for a user and the time where the file can actually be accessed by the user on other clients.

SEE ALSO

chmod(2), **flock**(2)


TOC

NAME

chroot - change root directory

SYNOPSIS

#include <unistd.h>

int chroot(const char *****path));

DESCRIPTION

chroot changes the root directory to that specified in *path*. This directory will be used for path names beginning with /. The root directory is inherited by all children of the current process.

Only the super-user may change the root directory.

Note that this call does not change the current working directory, so that `.' can be outside the tree rooted at `/'.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

Depending on the file system, other errors can be returned. The more general errors are listed below:

EPERM The effective UID is not zero.

EFAULT *path* points outside your accessible address space.

ENAMETOOLONG
path is too long.

ENOENT The file does not exist.

ENOMEM Insufficient kernel memory was available.

ENOTDIR A component of *path* is not a directory.

EACCES Search permission is denied on a component of the path prefix.

ELOOP Too many symbolic links were encountered in resolving *path*.

EIO An I/O error occurred.

CONFORMING TO

SVr4, SVID, 4.4BSD, X/OPEN. This function is not part of POSIX.1. SVr4 documents additional EINTR, ENOLINK and EMULTIHOP error conditions. X/OPEN does not document EIO, ENOMEM or EFAULT error conditions. This interface is marked as legacy by X/OPEN.

SEE ALSO

chdir(2)


TOC

NAME

__clone - create a child process

SYNOPSIS

#include <sched.h>

int __clone(int (*fn***) (void *****arg***), void *****child_stack**

DESCRIPTION

__clone creates a new process like **fork**(2) does. Unlike **fork**(2), **__clone** allows the child process to share parts of its execution context with its parent process, such as the memory space, the table of file descriptors, and the table of signal handlers. The main use of **__clone** is to implement threads: multiple threads of control in a program that run concurrently in a shared memory space.

When the child process is created, it executes the function application *fn(arg)*. The *fn* argument is a pointer to a function that is called by the child process at the beginning of its execution. The *arg* argument is passed back to the *fn* function.

When the *fn(arg)* function application returns, the child process terminates. The integer returned by *fn* is the exit code for the child process. The child process may also terminate explicitely by calling **exit**(1) or after receiving a fatal signal.

The *child_stack* argument specifies the location of the stack

used by the child process. Since the child and parent processes may share memory, it is not possible in general for the child process to execute in the same stack as the parent process. The parent process must therefore set up memory space for the child stack and pass a pointer to this space to **__clone**. Stacks grow downwards on all processors that run Linux (except the HP PA processors), so *child_stack* usually points to the topmost address of the memory space set up for the child stack.

The low byte of *flags* contains the number of the signal sent to the parent when the child dies. *flags* may also be bitwise-or'ed with one or several of the following constants, in order to specify what is shared between the parent and child processes:

CLONE_VM

If **CLONE_VM** is set, the parent and the child processes run in the same memory space. In particular, memory writes performed by the parent process or by the child process are also visible in the other process. Moreover, any memory mapping or unmapping performed with **mmap**(2) or **munmap**(2) by the child or parent process also affects the other process.

If **CLONE_VM** is not set, the child process runs in a separate copy of the memory space of the parent at the time of **__clone**. Memory writes or file mapping/unmapping performed by one of the processes does not affect the other, as in the case of **fork**(2).

CLONE_FS

If **CLONE_FS** is set, the parent and the child processes share the same file system information. This includes the root of the file system, the current working directory, and the umask. Any call to **chroot**(2), **chdir**(2), or **umask**(2) performed by the parent or child process also takes effect in the other process.

If **CLONE_FS** is not set, the child process works on a copy of the file system information of the parent at the time of **__clone**. Calls to **chroot**(2),**chdir**(2),**umask**(2) performed later by one of the processes does not affect the other.

CLONE_FILES

If **CLONE_FILES** is set, the parent and the child processes share the same file descriptor table. File descriptors always refer to the same files in the parent and in the child process. Any file descriptor created by the parent process or by the child process is also valid in the other process. Similarly, if one of the processes closes a file descriptor, or changes its associated flags, the other process is also affected.

If **CLONE_FILES** is not set, the child process inherits a copy of all file descriptors opened in the parent process at the time of **__clone**. Operations on file descriptors performed later by one of the parent or child processes do not affect the other.

CLONE_SIGHAND

If **CLONE_SIGHAND** is set, the parent and the child processes share the same table of signal handlers. If the parent or child process calls **sigaction**(2) to change the behavior associated with a signal, the behavior is also changed in the other process as well. However, the parent and child processes still have distinct signal masks and sets of pending signals. So, one of them may block or unblock some signals using **sigprocmask**(2) without affecting the other process.

If **CLONE_SIGHAND** is not set, the child process inherits a copy of the signal handlers of its parent at the time **__clone** is called. Calls to **sigaction**(2) performed later by one of the processes have no effect on the other process.

CLONE_PID

If **CLONE_PID** is set, the child process is created with the same process ID as its parent process.

If **CLONE_PID** is not set, the child process possesses a unique process ID, distinct from that of its parent.

RETURN VALUE

On success, the PID of the child process is returned in the parent's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and *errno* will be set appropriately.

ERRORS

EAGAIN
Too many processes are already running.

ENOMEM
__clone cannot allocate sufficient memory to allocate a task structure for the child, or to copy those parts of the parent's context that need to be copied.

BUGS

As of version 2.1.97 of the kernel, the **CLONE_PID** flag should not be used, since other parts of the kernel and most system software still assume that process IDs are unique.

There is no entry for **__clone** in libc version 5. libc 6 (a.k.a. glibc 2) provides **__clone** as described in this manual page.

CONFORMING TO

The **__clone** call is Linux-specific and should not be used in programs intended to be portable. For programming threaded applications (multiple threads of control in the same memory space), it is better to use a library implementing the POSIX 1003.1c thread API, such as the LinuxThreads library. See **pthread_create**(3thr).

This manual page corresponds to kernels 2.0.x and 2.1.x, and to glibc 2.0.x.

SEE ALSO

fork(2), **pthread_create**(3thr).

TOC

TOC

NAME

close - close a file descriptor

SYNOPSIS

#include <unistd.h>

int close(int *fd*));

DESCRIPTION

close closes a file descriptor, so that it no longer refers to any file and may be reused. Any locks held on the file it was associated with, and owned by the process, are removed (regardless of the file descriptor that was used to obtain the lock).

If *fd* is the last copy of a particular file descriptor the resources associated with it are freed; if the descriptor was the last reference to a file which has been removed using **unlink** the file is deleted.

RETURN VALUE

close returns zero on success, or -1 if an error occurred.

ERRORS

EBADF
fd isn't a valid open file descriptor.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, BSD 4.3. SVr4 documents an additional ENOLINK error condition.

NOTES

Not checking the return value of close is a common but nevertheless serious programming error. File system implementations which use techniques as ``write-behind'' to increase performance may lead to *write*(2) succeeding, although the data has not been written yet. The error status may be reported at a later write operation, but it is guaranteed to be reported on closing the file. Not checking the return value when closing the file may lead to silent loss of data. This can especially be observed with NFS and disk quotas.

SEE ALSO

open(2), **fcntl**(2), **shutdown**(2), **unlink**(2), **fclose**(3).

TOC

NAME

connect - initiate a connection on a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int
addrlen );
```

DESCRIPTION

The parameter *sockfd* is a socket. If it is of type **SOCK_DGRAM**, this call specifies the peer with which the socket is to be associated; this address is that to which datagrams are to be sent, and the only address from which datagrams are to be received. If the socket is of type **SOCK_STREAM** , this call attempts to make a connection to another socket. The other socket is specified by **serv_addr**, which is an address in the communications space of the socket. Each communications space interprets the **serv_addr**, parameter in its own way. Generally, stream sockets may successfully **connect** only once; datagram sockets may use **connect** multiple times to change their association. Datagram sockets may dissolve the association by connecting to an invalid address, such as a null address.

RETURN VALUE

If the connection or binding succeeds, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

The following are general socket errors only. There may be other domain-specific error codes.

EBADF Bad descriptor.

EFAULT The socket structure address is outside your address space.

ENOTSOCK
The descriptor is not associated with a socket.

EISCONN The socket is already connected.

ECONNREFUSED
Connection refused at server.

ETIMEDOUT
Timeout while attempting connection.

ENETUNREACH
Network is unreachable.

EADDRINUSE
Address is already in use.

EINPROGRESS
The socket is non-blocking and the connection cannot be completed immediately. It is possible to **select**(2) for completion by selecting the socket for writing. After **select** indicates writability, use **getsockopt**(2) to read the **SO_ERROR** option at level **SOL_SOCKET** to determine whether **connect** completed successfully (**SO_ERROR** is zero) or unsuccessfully

(**SO_ERROR** is one of the usual error codes listed above, explaining the reason for the failure).

EALREADY
The socket is non-blocking and a previous connection attempt has not yet been completed.

CONFORMING TO

SVr4, 4.4BSD (the **connect** function first appeared in BSD 4.2). SVr4 documents additional general error codes EADDRNOTAVAIL, EINVAL, EAFNOSUPPORT, EALREADY, EINTR, EPROTOTYPE, ENOSR. It also documents many additional error conditions not described here.

SEE ALSO

accept(2), **bind**(2), **listen**(2), **socket**(2), **getsockname**(2)

TOC

TOC

NAME

dup, dup2 - duplicate a file descriptor

SYNOPSIS

```
#include <unistd.h>

int dup(int oldfd));
int dup2(int oldfd, int newfd));
```

DESCRIPTION

dup and **dup2** create a copy of the file descriptor *oldfd*.

The old and new descriptors may be used interchangeably. They share locks, file position pointers and flags; for example, if the file position is modified by using **lseek** on one of the descriptors, the position is also changed for the other.

The two descriptors do not share the close-on-exec flag, however.

dup uses the lowest-numbered unused descriptor for the new descriptor.

dup2 makes *newfd* be the copy of *oldfd*, closing *newfd* first if necessary.

RETURN VALUE

dup and **dup2** return the new descriptor, or -1 if an error occurred (in which case, *errno* is set appropriately).

ERRORS

EBADF
oldfd isn't an open file descriptor, or *newfd* is out of the allowed range for file descriptors.

EMFILE
The process already has the maximum number of file descriptors open and tried to open a new one.

WARNING

The error returned by **dup2** is different to that returned by **fcntl(...,F_DUPFD,...)** when *newfd* is out of range. On some systems **dup2** also sometimes returns **EINVAL** like **F_DUPFD**.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, BSD 4.3. SVr4 documents additional EINTR and ENOLINK error conditions. POSIX.1 adds EINTR.

SEE ALSO

fcntl (2), **open** (2), **close** (2).

TOC

NAME

execve - execute program

SYNOPSIS

#include <unistd.h>

int execve (const char * *filename*, **char *const** *argv* **[], char *const** *envp*[]);

DESCRIPTION

execve() executes the program pointed to by *filename*. *filename* must be either a binary executable, or a script starting with a line of the form "**#!** *interpreter* [arg]". In the latter case, the interpreter must be a valid pathname for an executable which is not itself a script, which will be invoked as **interpreter** [arg] *filename*.

execve() does not return on success, and the text, data, bss, and stack of the calling process are overwritten by that of the program loaded. The program invoked inherits the calling process's PID, and any open file descriptors that are not set to close on exec. Signals pending on the parent process are cleared. Any signals set to be caught by the calling process are reset to their default behaviour.

If the current program is being ptraced, a **SIGTRAP** is sent to it after a successful **execve()**.

If the executable is an a.out dynamically-linked binary executable containing shared-library stubs, the Linux dynamic linker *ld.so*(8) is called at the start of execution to bring

needed shared libraries into core and link the executable with them.

If the executable is a dynamically-linked ELF executable, the interpreter named in the PT_INTERP segment is used to load the needed shared libraries. This interpreter is typically */lib/ld-linux.so*.**1** for binaries linked with the Linux libc version 5, or */lib/ld-linux.so*.**2** for binaries linked with the GNU libc version 2.

RETURN VALUE

On success, **execve()** does not return, on error -1 is returned, and *errno* is set appropriately.

ERRORS

EACCES The file or a script interpreter is not a regular file.

EACCES Execute permission is denied for the file or a script interpreter.

EACCES The file system is mounted *noexec*.

EPERM The file system is mounted *nosuid*, the user is not the superuser, and the file has an SUID or SGID bit set.

EPERM The process is being traced, the user is not the superuser and the file has an SUID or SGID bit set.

E2BIG The argument list is too big.

ENOEXEC An executable is not in a recognised format, is for the wrong architecture, or has some other format error that means it cannot be executed.

EFAULT *filename* points outside your accessible address space.

ENAMETOOLONG
filename is too long.

ENOENT The file *filename* or a script or ELF interpreter does not exist.

ENOMEM Insufficient kernel memory was available.

ENOTDIR A component of the path prefix of *filename* or a script or ELF interpreter is not a directory.

EACCES Search permission is denied on a component of the path prefix of *filename* or the name of a script interpreter.

ELOOP Too many symbolic links were encountered in resolving *filename* or the name of a script or ELF interpreter.

ETXTBUSY
Executable was open for writing by one or more processes.

EIO An I/O error occurred.

ENFILE The limit on the total number of files open on the system has been reached.

EMFILE The process has the maximum number of files open.

EINVAL An ELF executable had more than one PT_INTERP segment (i.e., tried to name more than one interpreter).

EISDIR An ELF interpreter was a directory.

ELIBBAD An ELF interpreter was not in a recognised format.

CONFORMING TO

SVr4, SVID, X/OPEN, BSD 4.3. POSIX does not document the #! behavior but is otherwise compatible. SVr4 documents additional error conditions EAGAIN, EINTR, ELIBACC, ENOLINK, EMULTIHOP; POSIX does not document ETXTBSY, EPERM, EFAULT, ELOOP, EIO, ENFILE, EMFILE, EINVAL, EISDIR or ELIBBAD error conditions.

NOTES

SUID and SGID processes can not be **ptrace()**d SUID or SGID.

A maximum line length of 127 characters is allowed for the first line in a #! executable shell script.

Linux ignores the SUID and SGID bits on scripts.

SEE ALSO

ld.so(8), **execl**(3), **fork**(2)

TOC

NAME

fcntl - manipulate file descriptor

SYNOPSIS

```
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fd, int cmd));
int fcntl(int fd, int cmd, long arg
```

DESCRIPTION

fcntl performs one of various miscellaneous operations on *fd*. The operation in question is determined by *cmd*:

F_DUPFD Makes *arg* be a copy of *fd*, closing *fd* first if necessary.

The same functionality can be more easily achieved by using **dup2**(2).

The old and new descriptors may be used interchangeably. They share locks, file position pointers and flags; for example, if the file position is modified by using **lseek** on one of the descriptors, the position is also changed for the other.

The two descriptors do not share the close-on-exec flag, however. The close-on-exec flag of the copy is off, meaning that it will be closed on exec.

On success, the new descriptor is returned.

F_GETFD Read the close-on-exec flag. If the low-order bit is 0, the file will remain open across **exec**, otherwise it will be closed.

F_SETFD Set the close-on-exec flag to the value specified by *arg* (only the least significant bit is used).

F_GETFL Read the descriptor's flags (all flags (as set by **open**(2)) are returned).

F_SETFL Set the descriptor's flags to the value specified by *arg*. Only **O_APPEND** and **O_NONBLOCK** may be set.

The flags are shared between copies (made with **dup** etc.) of the same file descriptor.

The flags and their semantics are described in **open**(2).

F_GETLK, F_SETLK and **F_SETLKW**
Manage discretionary file locks. The third argument *arg* is a pointer to a struct flock (that may be overwritten by this call).

F_GETLK Return the flock structure that prevents us from obtaining the lock, or set the **l_type** field of the lock to **F_UNLCK** if there is no obstruction.

F_SETLK The lock is set (when **l_type** is **F_RDLCK** or **F_WRLCK**) or cleared (when it is **F_UNLCK**). If the lock is held by someone else, this call returns -1 and sets *errno* to **EACCES** or **EAGAIN**.

F_SETLKW Like **F_SETLK**, but instead of returning an error we wait for the lock to be released. If a signal that is to be caught is received while *fcntl*() is waiting, it is interrupted and returns immediately (with return value -1 and *errno* set to EINTR).

F_GETOWN Get the process ID or process group currently receiving SIGIO and SIGURG signals for events on file descriptor *fd*. Process groups are returned as negative values.

F_SETOWN Set the process ID or process group that will receive SIGIO and SIGURG signals for events on file descriptor *fd*. Process groups are specified using negative values.

If you set the O_ASYNC status flag on a file descriptor (either by providing this flag with the *open* call, or by using the F_SETFL command of *fcntl)*, a SIGIO signal is sent whenever input or output becomes possible on that file descriptor. The process or process group to receive the signal can be selected by using the F_SETOWN command to the *fcntl* function. If the file descriptor is a socket, this also selects the recipient of SIGURG signals that are delivered when out-of-band data arrives on that socket. (SIGURG is sent in any situation where *select* would report the socket as having an "exceptional condition".) If the file descriptor corresponds to a terminal device, then SIGIO signals are sent to the foreground process group of the terminal.

The use of O_ASYNC, F_GETOWN, F_SETOWN is BSD-specific. POSIX has asynchronous I/O and the *aio_sigevent* structure to achieve similar things.

RETURN VALUE

For a successful call, the return value depends on the operation:

F_DUPFD The new descriptor.

F_GETFD Value of flag.

F_GETFL Value of flags.

F_GETOWN Value of descriptor owner.

F_SETFD, **F_SETFL**, **F_GETLK**, **F_SETLK**, **F_SETLKW** Some value dif-

ferent from -1.

On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EACCES Operation is prohibited by locks held by other processes.

EAGAIN Operation is prohibited because the file has been memory-mapped by another process.

EDEADLK It was detected that the specified F_SETLKW command would cause a deadlock.

EBADF *fd* is not an open file descriptor.

EINTR The F_SETLKW command was interrupted by a signal.

EINVAL For **F_DUPFD**, *arg* is negative or is greater than the maximum allowable value.

EMFILE For **F_DUPFD**, the process already has the maximum number of file descriptors open.

ENOLCK Too many segment locks open, lock table is full.

NOTES

The errors returned by **dup2** are different from those returned by **F_DUPFD**.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, BSD 4.3. Only the operations F_DUPFD, F_GETFD, F_SETFD, F_GETFL, F_SETFL, F_GETLK, F_SETLK and F_SETLKW are specified in POSIX.1; F_GETOWN and F_SETOWN are BSDisms not supported in SVr4. The flags legal for F_GETFL/F_SETFL are those supported by **open**(2) and vary between these systems; O_APPEND, O_NONBLOCK, O_RDONLY, and O_RDWR are specified in POSIX.1. SVr4 supports several other options and flags not documented here.
POSIX.1 documents an additional EINTR condition. SVr4 documents additional EFAULT, EINTR, EIO, ENOLINK and EOVERFLOW error conditions.

SEE ALSO

dup2(2), **open**(2), **socket**(2), **flock**(2)

TOC

TOC

NAME

fdatasync - synchronize a file's in-core data with that on disk

SYNOPSIS

```
#include <unistd.h>

#ifdef _POSIX_SYNCHRONIZED_IO

int fdatasync(int fd));

#endif
```

DESCRIPTION

fdatasync flushes all data buffers of a file to disk (before the system call returns). It resembles **fsync** but is not required to update the metadata such as access time.

Applications that access databases or log files often write a tiny data fragment (e.g., one line in a log file) and then call **fsync** immediately in order to ensure that the written data is physically stored on the harddisk. Unfortunately, **fsync** will always initiate two write operations: one for the newly written data and another one in order to update the modification time stored in the inode. If the modification time is not a part of the transaction concept **fdatasync** can be used to avoid unnecessary inode disk write operations.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF *fd* is not a valid file descriptor open for writing.

EROFS, **EINVAL**
fd is bound to a special file which does not support synchronization.

EIO An error occurred during synchronization.

BUGS

Currently (Linux 2.0.23) **fdatasync** is equivalent to **fsync**.

CONFORMING TO

POSIX1b (formerly POSIX.4)

SEE ALSO

fsync(2), B.O. Gallmeister, POSIX.4, O'Reilly, pp. 220-223 and 343.

NAME

flock - apply or remove an advisory lock on an open file

SYNOPSIS

```
#include <sys/file.h>

int flock(int fd, int operation)
```

DESCRIPTION

Apply or remove an advisory lock on an open file. The file is specified by *fd*. Valid operations are given below:

LOCK_SH	Shared lock. More than one process may hold a shared lock for a given file at a given time.
LOCK_EX	Exclusive lock. Only one process may hold an exclusive lock for a given file at a given time.
LOCK_UN	Unlock.
LOCK_NB	Don't block when locking. May be specified (by *or*'ing) along with one of the other operations.

A single file may not simultaneously have both shared and exclusive locks.

A file is locked (i.e., the inode), *not* the file descriptor. So, **dup**(2) and **fork**(2) do not create multiple instances of a lock.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EWOULDBLOCK
The file is locked and the **LOCK_NB** flag was selected.

CONFORMING TO

4.4BSD (the **flock**(2) call first appeared in 4.2BSD).

SEE ALSO

open(2), **close**(2), **dup**(2), **execve**(2), **fcntl**(2), **fork**(2). There are also *locks.txt* and *mandatory.txt* in */usr/src/linux/Documentation*.

NAME

fork, vfork - create a child process

SYNOPSIS

```
#include <unistd.h>

pid_t fork(void);
pid_t vfork(void);
```

DESCRIPTION

fork creates a child process that differs from the parent process only in its PID and PPID, and in the fact that resource utilizations are set to 0. File locks and pending signals are not inherited.

Under Linux, **fork** is implemented using copy-on-write pages, so the only penalty incurred by fork is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child.

RETURN VALUE

On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and *errno* will be set appropriately.

ERRORS

EAGAIN
fork cannot allocate sufficient memory to copy the parent's page tables and allocate a task structure for the child.

ENOMEM
fork failed to allocate the necessary kernel structures because memory is tight.

BUGS

Under Linux, **vfork** is merely an alias for **fork**.

CONFORMING TO

The **fork** call conforms to SVr4, SVID, POSIX, X/OPEN, BSD 4.3.

SEE ALSO

clone(2), **execve**(2), **wait**(2)

NAME

fsync - synchronize a file's complete in-core state with that on disk

SYNOPSIS

```
#include <unistd.h>

int fsync(int fd));
```

DESCRIPTION

fsync copies all in-core parts of a file to disk.

In some applications, **fdatasync** is a more efficient alternative to **fsync**.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF *fd* is not a valid file descriptor open for writing.

EROFS, EINVAL
fd is bound to a special file which does not support synchronization.

EIO An error occurred during synchronization.

CONFORMING TO

POSIX.1b (formerly POSIX.4)

SEE ALSO

bdflush(2), **fdatasync**(2), **sync**(2), **update**(8), **sync**(8)

NAME

getdents - get directory entries

SYNOPSIS

```
#include <unistd.h>
#include <linux/dirent.h>
#include <linux/unistd.h>

_syscall3(int, getdents, uint, fd, struct dirent

int getdents(unsigned int fd, struct dirent *dirp, unsigned int count
```

DESCRIPTION

getdents reads several *dirent* structures from the directory pointed at by *fd* into the memory area pointed to by *dirp*. The parameter *count* is the size of the memory area.

The *dirent* structure is declared as follows:

```
struct dirent
{
    long d_ino;                 /* inode number */
    off_t d_off;                /* offset to next dirent */
    unsigned short d_reclen;    /* length of this dirent */
    char d_name [NAME_MAX+1];   /* file name (null-terminated) */
}
```

d_ino is an inode number. *d_off* is the distance from the start of the directory to the start of the next *dirent*. *d_reclen* is the size of this entire *dirent*. *d_name* is a null-terminated file name.

This call supersedes **readdir**(2).

TOC

RETURN VALUE

On success, the number of bytes read is returned. On end of directory, 0 is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF Invalid file descriptor *fd*.

EFAULT Argument points outside the calling process's address space.

EINVAL Result buffer is too small.

ENOENT No such directory.

ENOTDIR File descriptor does not refer to a directory.

CONFORMING TO

SVr4, SVID. SVr4 documents additional ENOLINK, EIO error conditions.

SEE ALSO

readdir(2), **readdir**(3)


TOC

NAME

getdomainname, setdomainname - get/set domain name

SYNOPSIS

```
#include <unistd.h>

int getdomainname(char *name, size_t len));
int setdomainname(const char *name, size_t len));
```

DESCRIPTION

These functions are used to access or to change the domain name of the current processor.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
For **getdomainname**, *name* points to **NULL** or *name* is longer than *len*.

EPERM
For **setdomainname**, the caller was not the superuser.

EINVAL
For **setdomainname**, *len* was too long.

CONFORMING TO

POSIX does not specify these calls.

BUGS

getdomainname is not compliant with other implementations, since they always return *len* bytes, even if *name* is longer. Linux, however, returns **EINVAL** in this case (as of DLL 4.4.1 libraries).

NOTES

Under Linux, **getdomainname** is implemented at the library level by calling **uname**(2).

SEE ALSO

gethostname(2), **sethostname**(2), **uname**(2)

TOC

NAME

getdtablesize - get descriptor table size

SYNOPSIS

```
#include <unistd.h>

int getdtablesize(void);
```

DESCRIPTION

getdtablesize returns the maximum number of files a process can have open.

NOTES

getdtablesize is implemented as a library function in DLL 4.4.1. This function returns **OPEN_MAX** (set to 256 in Linux 2.0.23) if **OPEN_MAX** was defined when the library was compiled. Otherwise, -1 is returned, and *errno* is set to **ENOSYS**.

CONFORMING TO

SVr4, 4.4BSD (the **getdtablesize** function first appeared in

BSD 4.2).

SEE ALSO

close(2), **dup**(2), **open**(2)

NAME

getgid, getegid - get group identity

SYNOPSIS

```
#include <unistd.h>
#include <sys/types.h>

gid_t getgid(void);
gid_t getegid(void);
```

DESCRIPTION

getgid returns the real group ID of the current process.

getegid returns the effective group ID of the current process.

The real ID corresponds to the ID of the calling process. The effective ID corresponds to the set ID bit on the file being executed.

ERRORS

These functions are always successful.

CONFORMING TO

POSIX, BSD 4.3

SEE ALSO

setregid(2), **setgid**(2)

TOC

NAME

getgroups, setgroups - get/set list of supplementary group IDs

SYNOPSIS

```
#include <unistd.h>

int getgroups(int size, gid_t list[]);


#define __USE_BSD
#include <grp.h>

int setgroups(size_t size, const gid_t *list);
```

DESCRIPTION

getgroups
Up to *size* supplementary groups are returned in *list*. If *size* is zero, *list* is not modified, but the total number of supplementary groups for the process is returned.

setgroups
Sets the supplementary groups for the process. Only the super-user may use this function.

RETURN VALUE

getgroups
On success, the number of supplementary group IDs is returned. On error, -1 is returned, and *errno* is set appropriately.

setgroups
On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EFAULT
list has an invalid address.

EPERM
For **setgroups**, the user is not the super-user.

EINVAL
For **setgroups**, *size* is greater than **NGROUPS** (32 for Linux 2.0.32). For **getgroups**, *size* is less than the number of supplementary group IDs, but is not zero.

CONFORMING TO

SVr4, SVID (issue 4 only; these calls were not present in SVr3), X/OPEN, 4.3BSD. The **getgroups** function is in POSIX.1. Since **setgroups** requires privilege, it is not covered by POSIX.1.

BUGS

The **__USE_BSD** flag probably shouldn't be required for **setgroups**.

SEE ALSO

initgroups(3), **getgid**(2), **setgid**(2)


TOC

NAME

gethostid, sethostid - get or set the unique identifier of the current host

SYNOPSIS

```
#include <unistd.h>

long int gethostid(void);
int sethostid(long int hostid));
```

DESCRIPTION

Get or set a unique 32-bit identifier for the current machine. The 32-bit identifier is intended to be unique among all UNIX systems in existence. This normally resembles the Internet address for the local machine, as returned by **gethostbyname**(3), and thus usually never needs to be set.

The **sethostid** call is restricted to the superuser.

The *hostid* argument is stored in the file */etc/hostid*.

RETURN VALUES

gethostid returns the 32-bit identifier for the current host as set by **sethostid**(2).

CONFORMING TO

4.2BSD. These functions were dropped in 4.4BSD. POSIX.1 does not define these functions, but ISO/IEC 9945-1:1990 mentions them in B.4.4.1. SVr4 includes **gethostid** but not **sethostid**.

FILES

/etc/hostid

SEE ALSO

hostid(1), **gethostbyname**(3)

NAME

gethostname, sethostname - get/set host name

SYNOPSIS

```
#include <unistd.h>

int gethostname(char *name, size_t len));
int sethostname(const char *name, size_t len));
```

DESCRIPTION

These functions are used to access or to change the host name of the current processor.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
len is negative or, for **sethostname**, *len* is larger than the maximum allowed size, or, for **gethostname** on

Linux/i386, *len* is smaller than the actual size.

EPERM
For **sethostname**, the caller was not the superuser.

EFAULT
name is an invalid address.

CONFORMING TO

SVr4, 4.4BSD (this function first appeared in 4.2BSD). POSIX.1 does not define these functions, but ISO/IEC 9945-1:1990 mentions them in B.4.4.1.

BUGS

Some other implementations of **gethostname** successfully return *len* bytes even if *name* is longer. Linux/Alpha complies with this behaviour. Linux/i386, however, returns **EINVAL** in this case (as of DLL 4.6.27 libraries).

NOTES

Under Linux/Alpha, **gethostname** is a system call. Under Linux/i386, **gethostname** is implemented at the library level by calling **uname**(2).

SEE ALSO

getdomainname(2), **setdomainname**(2), **uname**(2)

TOC

TOC

NAME

getitimer, setitimer - get or set value of an interval timer

SYNOPSIS

```
#include <sys/time.h>

int getitimer(int which, struct itimerval *value));
int setitimer(int which, const struct itimerval *value,
     struct itimerval *ovalue));
```

DESCRIPTION

The system provides each process with three interval timers, each decrementing in a distinct time domain. When any timer expires, a signal is sent to the process, and the timer (potentially) restarts.

ITIMER_REAL decrements in real time, and delivers **SIGALRM** upon expiration.

ITIMER_VIRTUAL decrements only when the process is executing, and delivers **SIGVTALRM** upon expiration.

ITIMER_PROF decrements both when the process executes and when the system is executing on behalf of the process. Coupled with **ITIMER_VIRTUAL**, this timer is usually used to profile the time spent by the application in user and kernel space. **SIGPROF** is delivered upon expiration.

Timer values are defined by the following structures:

```
     struct itimerval {
```


```
        struct timeval it_interval; /* next value */
        struct timeval it_value;    /* current value */
    };
    struct timeval {
        long tv_sec;                /* seconds */
        long tv_usec;               /* microseconds */
    };
```

Getitimer(2) fills the structure indicated by *value* with the current setting for the timer indicated by *which* (one of **ITIMER_REAL**, **ITIMER_VIRTUAL**, or **ITIMER_PROF**). The element **it_value** is set to the amount of time remaining on the timer, or zero if the timer is disabled. Similarly, **it_interval** is set to the reset value. **Setitimer**(2) sets the indicated timer to the value in *value*. If *ovalue* is nonzero, the old value of the timer is stored there.

Timers decrement from *it_value* to zero, generate a signal, and reset to *it_interval*. A timer which is set to zero (*it_value* is zero or the timer expires and *it_interval* is zero) stops.
Both *tv_sec* and *tv_usec* are significant in determining the duration of a timer.

Timers will never expire before the requested time, instead expiring some short, constant time afterwards, dependent on the system timer resolution (currently 10ms). Upon expiration, a signal will be generated and the timer reset. If the timer expires while the process is active (always true for **ITIMER_VIRT**) the signal will be delivered immediately when generated. Otherwise the delivery will be offset by a small time dependent on the system loading.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EFAULT *value* or *ovalue* are not valid pointers.

EINVAL *which* is not one of **ITIMER_REAL**, **ITIMER_VIRT**, or **ITIMER_PROF**.

CONFORMING TO

SVr4, 4.4BSD (This call first appeared in 4.2BSD).

SEE ALSO

gettimeofday(2), **sigaction**(2), **signal**(2).

BUGS

Under Linux, the generation and delivery of a signal are distinct, and there each signal is permitted only one outstanding event. It's therefore conceivable that under pathologically heavy loading, **ITIMER_REAL** will expire before the signal from a previous expiration has been delivered. The second signal in such an event will be lost.


TOC

NAME

getpagesize - get system page size

SYNOPSIS

```
#include <unistd.h>

size_t getpagesize(void);
```

DESCRIPTION

Return the number of bytes in a page. This is the system's page size, which is not necessarily the same as the hardware page size.

NOTES

getpagesize is implemented as a library function in DLL 4.4.1. Depending on what is defined when the library is compiled, this function returns **EXEC_PAGESIZE** (set to 4096 in Linux 0.99.11), **NBPG** (set to 4096 in Linux 0.99.11), or **NBPC** (not defined in Linux 0.99.11 or DLL 4.4.1 libraries).

CONFORMING TO

SVr4, 4.4BSD (this call first appeared in 4.2BSD).

SEE ALSO

sbrk(2)

NAME

getpeername - get name of connected peer

SYNOPSIS

#include <sys/socket.h>

int getpeername(int s, struct sockaddr *_name_, int *_namelen_

DESCRIPTION

Getpeername returns the name of the peer connected to socket **s**. The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF The argument **s** is not a valid descriptor.

ENOTSOCK
The argument **s** is a file, not a socket.

ENOTCONN
The socket is not connected.

ENOBUFS Insufficient resources were available in the system to perform the operation.

EFAULT The *name* parameter points to memory not in a valid part of the process address space.

CONFORMING TO

SVr4, 4.4BSD (the **getpeername** function call first appeared in 4.2BSD).

SEE ALSO

accept(2), **bind**(2), **getsockname**(2)

NAME

getpid, getppid - get process identification

SYNOPSIS

```
#include <unistd.h>

pid_t getpid(void);
pid_t getppid(void);
```

DESCRIPTION

getpid returns the process ID of the current process. (This is often used by routines that generate unique temporary file names.)

getppid returns the process ID of the parent of the current process.

CONFORMING TO

POSIX, BSD 4.3, SVID

SEE ALSO

exec(3), **fork**(2), **kill**(2), **mkstemp**(3), **tmpnam**(3), **tempnam**(3), **tmpfile**(3)

NAME

getpriority, setpriority - get/set program scheduling priority

SYNOPSIS

```
#include <sys/time.h>
#include <sys/resource.h>

int getpriority(int which, int who));
int setpriority(int which, int who, int prio
```

DESCRIPTION

The scheduling priority of the process, process group, or user, as indicated by *which* and *who* is obtained with the **getpriority** call and set with the **setpriority** call. *Which* is one of **PRIO_PROCESS**, **PRIO_PGRP**, or **PRIO_USER**, and *who* is interpreted relative to *which* (a process identifier for **PRIO_PROCESS**, process group identifier for **PRIO_PGRP**, and a user ID for **PRIO_USER**). A zero value of *who* denotes the current process, process group, or user. *Prio* is a value in the range -20 to 20. The default priority is 0; lower priorities cause more favorable scheduling.

The **getpriority** call returns the highest priority (lowest numerical value) enjoyed by any of the specified processes. The **setpriority** call sets the priorities of all of the specified processes to the specified value. Only the super-user may lower priorities.

RETURN VALUES

Since **getpriority** can legitimately return the value -1, it is necessary to clear the external variable *errno* prior to the call, then check it afterwards to determine if a -1 is an error or a legitimate value. The **setpriority** call returns 0 if there is no error, or -1 if there is.

ERRORS

ESRCH No process was located using the *which* and *who* values specified.

EINVAL *Which* was not one of **PRIO_PROCESS**, **PRIO_PGRP**, or **PRIO_USER**.

In addition to the errors indicated above, **setpriority** will fail if:

EPERM
A process was located, but neither its effective nor real user ID matched the effective user ID of the caller.

EACCES
A non super-user attempted to lower a process priority.

CONFORMING TO

SVr4, 4.4BSD (these function calls first appeared in 4.2BSD).

SEE ALSO

TOC

nice(1), **fork**(2), **renice**(8)

TOC

NAME

getresuid, getresgid - get real, effective and saved user or group ID

SYNOPSIS

```
#include <unistd.h>

int getresuid(uid_t *ruid, uid_t *euid, uid_t *suid
int getresgid(gid_t *rgid, gid_t *egid, gid_t *sgid
```

DESCRIPTION

getresuid and **getresgid** (both introduced in Linux 2.1.44) get the real, effective and saved user ID's (resp. group ID's) of the current process.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EFAULT
One of the arguments specified an address outside the calling program's address space.

CONFORMING TO

This call is Linux-specific.

SEE ALSO

getuid(2), **setuid**(2), **getreuid**(2), **setreuid**(2), **setresuid**(2)

TOC

NAME

getrlimit, getrusage, setrlimit - get/set resource limits and usage

SYNOPSIS

```
#include <sys/time.h>
#include <sys/resource.h>
#include <unistd.h>

int getrlimit (int resource, struct rlimit *rlim));
int getrusage (int who, struct rusage *usage));
int setrlimit (int resource, const struct rlimit *rlim));
```

DESCRIPTION

getrlimit and **setrlimit** get and set resource limits respectively. *resource* should be one of:

```
RLIMIT_CPU     /* CPU time in seconds */
RLIMIT_FSIZE   /* Maximum filesize */
RLIMIT_DATA    /* max data size */
RLIMIT_STACK   /* max stack size */
RLIMIT_CORE    /* max core file size */
RLIMIT_RSS     /* max resident set size */
RLIMIT_NPROC   /* max number of processes */
RLIMIT_NOFILE  /* max number of open files */
RLIMIT_MEMLOCK /* max locked-in-memory address space*/
```

A resource may unlimited if you set the limit to **RLIM_INFINITY**. **RLIMIT_OFILE** is the BSD name for **RLIMIT_NOFILE**.

The **rlimit** structure is defined as follows :


```
struct rlimit
{
     int  rlim_cur;
     int  rlim_max;
};
```

getrusage returns the current resource usages, for a *who* of either **RUSAGE_SELF** or **RUSAGE_CHILDREN.**

```
struct rusage
{
     struct timeval ru_utime; /* user time used */
     struct timeval ru_stime; /* system time used */
     long ru_maxrss;          /* maximum resident set size */
     long ru_ixrss;      /* integral shared memory size */
     long ru_idrss;      /* integral unshared data size */
     long ru_isrss;      /* integral unshared stack size */
     long ru_minflt;          /* page reclaims */
     long ru_majflt;          /* page faults */
     long ru_nswap;      /* swaps */
     long ru_inblock;         /* block input operations */
     long ru_oublock;         /* block output operations */
     long ru_msgsnd;          /* messages sent */
     long ru_msgrcv;          /* messages received */
     long ru_nsignals;        /* signals received */
     long ru_nvcsw;      /* voluntary context switches */
     long ru_nivcsw;          /* involuntary context switches */
};
```

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EFAULT
rlim or *usage* points outside the accessible address space.

EINVAL

getrlimit or **setrlimit** is called with a bad *resource*, or **getrusage** is called with a bad *who*.

EPERM

A non-superuser tries to use **setrlimit()** to increase the soft or hard limit above the current hard limit, or a superuser tries to increase RLIMIT_NOFILE above the current kernel maximum.

CONFORMING TO

SVr4, BSD 4.3

SEE ALSO

ulimit(2), **quotactl**(2)

TOC

NAME

getsid - get session ID

SYNOPSIS

```
#include <unistd.h>

pid_t getsid(pid_t pid));
```

DESCRIPTION

getsid(0) returns the session ID of the calling process. **getsid(p)** returns the session ID of the process with process ID **p**.

ERRORS

On error, -1 will be returned. The only error which can happen is **ESRCH**, when no process with process ID **p** was found.

CONFORMING TO

SVr4, which documents an additional EPERM error condition.

SEE ALSO

setsid(2).

NAME

getsockname - get socket name

SYNOPSIS

#include <sys/socket.h>

int getsockname(int s , struct sockaddr * name , int * namelen

DESCRIPTION

Getsockname returns the current *name* for the specified socket. The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes).

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately. A 0 is returned if the call succeeds, -1 if it fails.

ERRORS

EBADF The argument **s** is not a valid descriptor.

ENOTSOCK
The argument **s** is a file, not a socket.

ENOBUFS Insufficient resources were available in the system to perform the operation.

EFAULT The *name* parameter points to memory not in a valid part of the process address space.

CONFORMING TO

SVr4, 4.4BSD (the **getsockname** function call appeared in 4.2BSD). SVr4 documents additional ENOMEM and ENOSR error codes.

SEE ALSO

bind(2), **socket**(2)

TOC

NAME

getsockopt, setsockopt - get and set options on sockets

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(int s, int level, int optname  void  *optval,
int *optlen));

int setsockopt(int s, int  level,  int  optname  const  void
*optval, int optlen));
```

DESCRIPTION

Getsockopt and **setsockopt** manipulate the *options* associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost **socket** level.

When manipulating socket options the level at which the option resides and the name of the option must be specified. To manipulate options at the socket level, *level* is specified as **SOL_SOCKET**. To manipulate options at any other level the protocol number of the appropriate protocol controlling the option is supplied. For example, to indicate that an option is to be interpreted by the **TCP** protocol, *level* should be set to the protocol number of **TCP**; see **getprotoent**(3).

The parameters *optval* and *optlen* are used to access option values for **setsockopt**. For **getsockopt** they identify a

buffer in which the value for the requested option(s) are to be returned. For **getsockopt**, *optlen* is a value-result parameter, initially containing the size of the buffer pointed to by *optval*, and modified on return to indicate the actual size of the value returned. If no option value is to be supplied or returned, *optval* may be NULL.

Optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <*sys/socket.h*> contains definitions for socket level options, described below. Options at other protocol levels vary in format and name; consult the appropriate entries in section 4 of the manual.

Most socket-level options utilize an *int* parameter for *optval*. For **setsockopt**, the parameter should be non-zero to enable a boolean option, or zero if the option is to be disabled. **SO_LINGER** uses a *struct linger* parameter, defined in <*linux/socket.h*>, which specifies the desired state of the option and the linger interval (see below). **SO_SNDTIMEO** and **SO_RCVTIMEO** use a *struct timeval* parameter, defined in <*sys/time.h*>.

The following options are recognized at the socket level. Except as noted, each may be examined with **getsockopt** and set with **setsockopt**.

SO_DEBUG
: enables recording of debugging information

SO_REUSEADDR
: enables local address reuse

SO_KEEPALIVE
: enables keep connections alive

SO_DONTROUTE
: enables routing bypass for outgoing messages

SO_LINGER
: linger on close if data present

SO_BROADCAST
: enables permission to transmit broadcast messages

SO_OOBINLINE

enables reception of out-of-band data in band

SO_SNDBUF
set buffer size for output

SO_RCVBUF
set buffer size for input

SO_SNDLOWAT
set minimum count for output

SO_RCVLOWAT
set minimum count for input

SO_SNDTIMEO
get timeout value for output (get only)

SO_RCVTIMEO
get timeout value for input (get only)

SO_TYPE get the type of the socket (get only)

SO_ERROR
get and clear error on the socket (get only)

SO_DEBUG enables debugging in the underlying protocol modules. **SO_REUSEADDR** indicates that the rules used in validating addresses supplied in a **bind**(2) call should allow reuse of local addresses. **SO_KEEPALIVE** enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified via a **SIGPIPE** signal when attempting to send data. **SO_DONTROUTE** indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO_LINGER controls the action taken when unsent messages are queued on socket and a **close**(2) is performed. If the socket promises reliable delivery of data and **SO_LINGER** is set, the system will block the process on the **close** attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the **setsockopt** call when **SO_LINGER** is requested). If **SO_LINGER** is disabled and a

close is issued, the system will process the close in a manner that allows the process to continue as quickly as possible.

The *linger* structure is defined in <*linux/socket.h*> as follows:

```
struct linger {
        int  l_onoff;   /* Linger active */
        int  l_linger;  /* How long to linger for */
};
```

l_onoff indicates wether to linger or not. If it is set to 1 then **l_linger** contains the time in hundredths of seconds how long the process should linger to complete the **close**. If **l_onoff** is set to zero the process returns immediately.

The option **SO_BROADCAST** requests permission to send broadcast datagrams on the socket. Broadcast was a privileged operation in earlier versions of the system. With protocols that support out-of-band data, the **SO_OOBINLINE** option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with **recv** or **read** calls without the **MSG_OOB** flag. Some protocols always behave as if this option is set. **SO_SNDBUF** and **SO_RCVBUF** are options to adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections, or may be decreased to limit the possible backlog of incoming data. The system places an absolute limit on these values.

SO_SNDLOWAT is an option to set the minimum count for output operations. Most output operations process all of the data supplied by the call, delivering data to the protocol for transmission and blocking as necessary for flow control. Nonblocking output operations will process as much data as permitted subject to flow control without blocking, but will process no data if flow control does not allow the smaller of the low water mark value or the entire request to be processed. A **select**(2) operation testing the ability to write to a socket will return true only if the low water mark amount could be processed. The default value for **SO_SNDLOWAT** is set to a convenient size for network efficiency, often 1024.

SO_RCVLOWAT is an option to set the minimum count for input

operations. In general, receive calls will block until any (non-zero) amount of data is received, then return with smaller of the amount available or the amount requested. The default value for **SO_RCVLOWAT** is 1. If **SO_RCVLOWAT** is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. Receive calls may still return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different than that returned.

SO_SNDTIMEO is an option to get the timeout value for output operations. (It can be used with *getsockopt* only). It returns a *struct timeval* parameter with the number of seconds and microseconds used to limit waits for output operations to complete. If a send operation has blocked for this much time, it returns with a partial count or with the error **EWOULDBLOCK** if no data were sent. In the current implementation, this timer is restarted each time additional data are delivered to the protocol, implying that the limit applies to output portions ranging in size from the low water mark to the high water mark for output. **SO_RCVTIMEO** is an option to get the timeout value for input operations. (It can be used with *getsockopt* only). It returns a *struct timeval* parameter with the number of seconds and microseconds used to limit waits for input operations to complete. In the current implementation, this timer is restarted each time additional data are received by the protocol, and thus the limit is in effect an inactivity timer. If a receive operation has been blocked for this much time without receiving additional data, it returns with a short count or with the error **EWOULDBLOCK** if no data were received.

Finally, also **SO_TYPE** and **SO_ERROR** are options used only with *getsockopt*. **SO_TYPE** returns the type of the socket, such as **SOCK_STREAM**; it is useful for servers that inherit sockets on startup. **SO_ERROR** returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF The argument **s** is not a valid descriptor.

ENOTSOCK
The argument **s** is a file, not a socket.

ENOPROTOOPT
The option is unknown at the level indicated.

EFAULT The address pointed to by *optval* is not in a valid part of the process address space. For **getsockopt**, this error may also be returned if *optlen* is not in a valid part of the process address space.

CONFORMING TO

SVr4, 4.4BSD (these system calls first appeared in 4.2BSD). SVr4 documents additional ENOMEM and ENOSR error codes, but does not document the **SO_SNDLOWAT**, **SO_RCVLOWAT**, **SO_SNDTIMEO**, options

BUGS

Several of the socket options should be handled at lower levels of the system.

SEE ALSO

ioctl(2), **socket**(2), **getprotoent**(3),

TOC

NAME

gettimeofday, settimeofday - get / set time

SYNOPSIS

```
#include <sys/time.h>
#include <unistd.h>

int gettimeofday(struct timeval *tv, struct timezone *tz));
int settimeofday(const struct timeval  *tv  ,  const  struct
timezone *tz));
```

DESCRIPTION

gettimeofday and **settimeofday** can set the time as well as a timezone. *tv* is a **timeval** struct, as specified in /usr/include/sys/time.h:

```
struct timeval {
        long tv_sec;        /* seconds */
        long tv_usec;  /* microseconds */
};
```

and *tz* is a **timezone** :

```
struct timezone {
        int  tz_minuteswest; /* minutes W of Greenwich */
        int  tz_dsttime;     /* type of dst correction */
};
```

The use of the timezone struct is obsolete; the *tz_dsttime* field has never been used under Linux - it has not been and will not be supported by libc or glibc. Each and every

occurrence of this field in the kernel source (other than the declaration) is a bug. Thus, the following is purely of historic interest.

The field *tz_dsttime* contains a symbolic constant (values are given below) that indicates in which part of the year Daylight Saving Time is in force. (Note: its value is constant throughout the year - it does not indicate that DST is in force, it just selects an algorithm.) The daylight saving time algorithms defined are as follows :

```
DST_NONE     /* not on dst */
DST_USA      /* USA style dst */
DST_AUST     /* Australian style dst */
DST_WET      /* Western European dst */
DST_MET      /* Middle European dst */
DST_EET      /* Eastern European dst */
DST_CAN      /* Canada */
DST_GB       /* Great Britain and Eire */
DST_RUM      /* Rumania */
DST_TUR      /* Turkey */
DST_AUSTALT  /* Australian style with shift in 1986 */
```

Of course it turned out that the period in which Daylight Saving Time is in force cannot be given by a simple algorithm, one per country; indeed, this period is determined by unpredictable political decisions. So this method of representing time zones has been abandoned. Under Linux, in a call to **settimeofday** the *tz_dsttime* field should be zero.

Under Linux there is some peculiar `warp clock' semantics associated to the **settimeofday** system call if on the very first call (after booting) that has a non-NULL *tz* argument, the *tv* argument is NULL and the *tz_minuteswest* field is nonzero. In such a case it is assumed that the CMOS clock is on local time, and that it has to be incremented by this amount to get UTC system time. No doubt it is a bad idea to use this feature.

The following macros are defined to operate on a struct timeval :

```
#define       timerisset(tvp)\
        ((tvp)->tv_sec || (tvp)->tv_usec)
#define       timercmp(tvp, uvp, cmp)\
        ((tvp)->tv_sec cmp (uvp)->tv_sec ||\
        (tvp)->tv_sec == (uvp)->tv_sec &&\
```


```
        (tvp)->tv_usec cmp (uvp)->tv_usec)
#define       timerclear(tvp)\
        ((tvp)->tv_sec = (tvp)->tv_usec = 0)
```

If either *tv* or *tz* is null, the corresponding structure is not set or returned.

Only the super user may use **settimeofday**.

RETURN VALUES

gettimeofday and **settimeofday** return 0 for success, or -1 for failure (in which case *errno* is set appropriately).

ERRORS

EPERM
settimeofday is called by someone other than the superuser.

EINVAL
Timezone (or something else) is invalid.

EFAULT
One of *tv* or *tz* pointed outside your accessible address space.

CONFORMING TO

SVr4, BSD 4.3

SEE ALSO

date(1), **adjtimex**(2), **time**(2),

TOC

NAME

getuid, geteuid - get user identity

SYNOPSIS

```
#include <unistd.h>
#include <sys/types.h>

uid_t getuid(void);
uid_t geteuid(void);
```

DESCRIPTION

getuid returns the real user ID of the current process.

geteuid returns the effective user ID of the current process.

The real ID corresponds to the ID of the calling process. The effective ID corresponds to the set ID bit on the file being executed.

ERRORS

These functions are always successful.

CONFORMING TO

POSIX, BSD 4.3.

SEE ALSO

setreuid(2), **setuid**(2)

NAME

idle - make process 0 idle

SYNOPSIS

```
#include <unistd.h>

void idle(void);
```

DESCRIPTION

idle is an internal system call used during bootstrap. It marks the process's pages as swappable, lowers its priority, and enters the main scheduling loop. **idle** never returns.

Only process 0 may call **idle**. Any user process, even a process with super-user permission, will receive **EPERM**.

RETURN VALUE

idle never returns for process 0, and always returns -1 for a user process.

ERRORS

EPERM Always, for a user process.

CONFORMING TO

This function is Linux-specific, and should not be used in programs intended to be portable.

NAME

intro - Introduction to system calls

DESCRIPTION

This chapter describes the Linux system calls. For a list of the 164 syscalls present in Linux 2.0, see syscalls(2).

Calling Directly

In most cases, it is unnecessary to invoke a system call directly, but there are times when the Standard C library does not implement a nice function call for you.

Synopsis

#include <linux/unistd.h>

A _syscall macro

desired system call

Setup

The important thing to know about a system call is its prototype. You need to know how many arguments, their types, and the function return type. There are six macros that make the actual call into the system easier. They have the form:

_syscall*X*(*type*,*name*

where **X** is 05, which are the number of arguments taken by the system call

type is the return type of the system call

name is the name of the system call

typeN is the Nth argument's type

argN is the name of the Nth argument

These macros create a function called *name* with the arguments you specify. Once you include the _syscall() in your source file, you call the system call by *name*.

EXAMPLE

```
#include <stdio.h>
#include <linux/unistd.h>     /* for _syscallX macros/related stuff */
#include <linux/kernel.h>     /* for struct sysinfo */

_syscall1(int, sysinfo, struct sysinfo *, info);

/* Note: if you copy directly from the nroff source, remember to
REMOVE the extra backslashes in the printf statement. */

int main(void)
{
     struct sysinfo s_info;
     int error;

     error = sysinfo(&s_info);
     printf("code error = %d\n", error);
        printf("Uptime = %ds\nLoad: 1 min %d / 5 min %d / 15 min %d\n"
                "RAM: total %d / free %d / shared %d\n"
                "Memory in buffers = %d\nSwap: total %d / free %d\n"
                "Number of processes = %d\n",
          s_info.uptime, s_info.loads[0],
          s_info.loads[1], s_info.loads[2],
          s_info.totalram, s_info.freeram,
          s_info.sharedram, s_info.bufferram,
          s_info.totalswap, s_info.freeswap,
          s_info.procs);
     return(0);
}
```

Sample Output

```
code error = 0
uptime = 502034s
```


TOC

```
Load: 1 min 13376 / 5 min 5504 / 15 min 1152
RAM: total 15343616 / free 827392 / shared 8237056
Memory in buffers = 5066752
Swap: total 27881472 / free 24698880
Number of processes = 40
```

NOTES

The _syscall() macros DO NOT produce a prototype. You may have to create one, especially for C++ users.

System calls are not required to return only positive or negative error codes. You need to read the source to be sure how it will return errors. Usually, it is the negative of a standard error code, e.g., -**EPERM**. The _syscall() macros will return the result **r** of the system call when **r** is nonnegative, but will return -1 and set the variable *errno* to -**r** when **r** is negative.

Some system calls, such as **mmap**, require more than five arguments. These are handled by pushing the arguments on the stack and passing a pointer to the block of arguments.

When defining a system call, the argument types MUST be passed by-value or by-pointer (for aggregates like structs).

CONFORMING TO

Certain codes are used to indicate Unix variants and standards to which calls in the section conform. These are:

SVr4 System V Release 4 Unix, as described in the "Programmer's Reference Manual: Operating System API (Intel processors)" (Prentice-Hall 1992, ISBN 0-13-951294-2)

SVID System V Interface Definition, as described in "The System V Interface Definition, Fourth Edition", available at ftp://ftp.fpk.novell.com/pub/unix-standards/svid in Postscript files.

POSIX.1
: IEEE 1003.1-1990 part 1, aka ISO/IEC 9945-1:1990s, aka "IEEE Portable Operating System Interface for Computing Environments", as elucidated in Donald Lewine's "POSIX Programmer's Guide" (O'Reilly & Associates, Inc., 1991, ISBN 0-937175-73-0.

POSIX.1b
: IEEE Std 1003.1b-1993 (POSIX.1b standard) describing real-time facilities for portable operating systems, aka ISO/IEC 9945-1:1996, as elucidated in "Programming for the real world - POSIX.4" by Bill O. Gallmeister (O'Reilly & Associates, Inc. ISBN 1-56592-074-0).

4.3BSD/4.4BSD
: The 4.3 and 4.4 distributions of Berkeley Unix. 4.4BSD was upward-compatible from 4.3.

V7
: Version 7, the ancestral Unix from Bell Labs.

FILES

/usr/include/linux/unistd.h

AUTHORS

Look at the source header of the manual page for the author(s) and copyright conditions. Note that these can be different from page to page!

TOC

NAME

ioctl - control device

SYNOPSIS

#include <sys/ioctl.h>

int ioctl(int d, int *request*, **...)**

[The "third" argument is traditionally **char** ***argp**, and will be so named for this discussion.]

DESCRIPTION

The **ioctl** function manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g. terminals) may be controlled with **ioctl** requests. The argument **d** must be an open file descriptor.

An ioctl *request* has encoded in it whether the argument is an *in* parameter or *out* parameter, and the size of the argument *argp* in bytes. Macros and defines used in specifying an ioctl *request* are located in the file <*sys/ioctl.h*>.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF **d** is not a valid descriptor.

ENOTTY **d** is not associated with a character special device.

ENOTTY The specified request does not apply to the kind of object that the descriptor **d** references.

EINVAL *Request* or *argp* is not valid.

CONFORMING TO

No single standard. Arguments, returns, and semantics of **ioctl**(2) vary according to the device driver in question (the call is used as a catch-all for operations that don't cleanly fit the Unix stream I/O model). See **ioctl_list**(2) for a list of many of the known **ioctl** calls. The **ioctl** function call appeared in Version 7 AT&T Unix.

SEE ALSO

execve(2), **fcntl**(2), **mt**(4),

An argument type of 'const struct foo *' means the argument is input to the kernel. 'struct foo *' means the kernel outputs the argument. If the kernel uses the argument for both input and output, this is marked with // I-O.

Some ioctls take more arguments or return more values than a single structure. These are marked // MORE and documented further in a separate section.

This list is incomplete. It does not include:

-- Ioctls defined internal to the kernel ('scsi_ioctl.h').
-- Ioctls defined in modules distributed separately from the kernel.

And, of course, I may have errors and omissions.

Please e-mail changes and comments to <mec@duracef.shout.net>. I am particularly interested in loadable modules which define their own ioctls. If you know of such a module, tell me where I can ftp it, and I'll include its ioctls in my next release.

// Main table.

```
// <include/asm-i386/socket.h>
0x00008901  FIOSETOWN                   const int *
0x00008902  SIOCSPGRP                   const int *
0x00008903  FIOGETOWN                   int *
0x00008904  SIOCGPGRP                   int *
0x00008905  SIOCATMARK                  int *
0x00008906  SIOCGSTAMP                  timeval *

// <include/asm-i386/termios.h>
0x00005401  TCGETS                      struct termios *
0x00005402  TCSETS                      const struct termios *
0x00005403  TCSETSW                     const struct termios *
0x00005404  TCSETSF                     const struct termios *
0x00005405  TCGETA                      struct termio *
0x00005406  TCSETA                      const struct termio *
0x00005407  TCSETAW                     const struct termio *
0x00005408  TCSETAF                     const struct termio *
0x00005409  TCSBRK                      int
0x0000540A  TCXONC                      int
0x0000540B  TCFLSH                      int
0x0000540C  TIOCEXCL                    void
0x0000540D  TIOCNXCL                    void
0x0000540E  TIOCSCTTY                   int
0x0000540F  TIOCGPGRP                   pid_t *
```


```
0x00005410  TIOCSPGRP                   const pid_t *
0x00005411  TIOCOUTQ                    int *
0x0000541F  TIOCSSERIAL                 const struct serial_struct *
0x00005420  TIOCPKT                     const int *
0x00005421  FIONBIO                     const int *
0x00005422  TIOCNOTTY                   void
0x00005423  TIOCSETD                    const int *
0x00005424  TIOCGETD                    int *
0x00005425  TCSBRKP                     int
0x00005426  TIOCTTYGSTRUCT              struct tty_struct *
0x00005450  FIONCLEX                    void
0x00005451  FIOCLEX                     void
0x00005452  FIOASYNC                    const int *
0x00005453  TIOCSERCONFIG               void
0x00005454  TIOCSERGWILD                int *
0x00005455  TIOCSERSWILD                const int *
0x00005456  TIOCGLCKTRMIOS              struct termios *
0x00005457  TIOCSLCKTRMIOS              const struct temios *
0x00005458  TIOCSERGSTRUCT              struct async_struct *
0x00005459  TIOCSERGETLSR               int *
0x0000545A  TIOCSERGETMULTI             struct serial_multiport_struct *
0x0000545B  TIOCSERSETMULTI             const struct serial_multiport_struct *

// <include/linux/ax25.h>
0x000089E0  SIOCAX25GETUID              const struct sockaddr_ax25 *
0x000089E1  SIOCAX25ADDUID              const struct sockaddr_ax25 *
0x000089E2  SIOCAX25DELUID              const struct sockaddr_ax25 *
0x000089E3  SIOCAX25NOUID               const int *
0x000089E4  SIOCAX25DIGCTL              const int *
0x000089E5  SIOCAX25GETPARMS            struct ax25_parms_struct *      // I-O
0x000089E6  SIOCAX25SETPARMS            const struct ax25_parms-struct *

// <include/linux/cdk.h>
0x00007314  STL_BINTR                   void
0x00007315  STL_BSTART                  void
0x00007316  STL_BSTOP                   void
0x00007317  STL_BRESET                  void

// <include/linux/cdrom.h>
0x00005301  CDROMPAUSE                  void
0x00005302  CDROMRESUME                 void
0x00005303  CDROMPLAYMSF                const struct cdrom_msf *
0x00005304  CDROMPLAYTRKIND             const struct cdrom_ti *
0x00005305  CDROMREADTOCHDR             struct cdrom_tochdr *
0x00005306  CDROMREADTOCENTRY           struct cdrom_tocentry *         // I-O
0x00005307  CDROMSTOP                   void
0x00005308  CDROMSTART                  void
0x00005309  CDROMEJECT                  void
0x0000530A  CDROMVOLCTRL                const struct cdrom_volctrl *
0x0000530B  CDROMSUBCHNL                struct cdrom_subchnl *          // I-O
0x0000530C  CDROMREADMODE2              const struct cdrom_msf *        // MORE
```


```
0x0000530D  CDROMREADMODE1              const struct cdrom_msf *        // |
0x0000530E  CDROMREADAUDIO              const struct cdrom_read_audio * // |
0x0000530F  CDROMEJECT_SW               int
0x00435902  CYGETTHRESH                 int *
0x00435903  CYSETTHRESH                 int
0x00435904  CYGETDEFTHRESH              int *
0x00435905  CYSETDEFTHRESH              int
0x00435906  CYGETTIMEOUT                int *
0x00435907  CYSETTIMEOUT                int
0x00435908  CYGETDEFTIMEOUT             int *
0x00435909  CYSETDEFTIMEOUT             int

// <include/linux/ext2_fs.h>
0x80046601  EXT2_IOC_GETFLAGS           int *
0x40046602  EXT2_IOC_SETFLAGS           const int *
0x80047601  EXT2_IOC_GETVERSION         int *
0x40047602  EXT2_IOC_SETVERSION         const int *

// <include/linux/fd.h>
0x00000000  FDCLRPRM                    void
0x00000001  FDSETPRM                    const struct floppy_struct *
0x00000002  FDDEFPRM                    const struct floppy_struct *
0x00000003  FDGETPRM                    struct floppy_struct *
0x00000004  FDMSGON                     void
0x00000005  FDMSGOFF                    void
0x00000006  FDFMTBEG                    void
0x00000007  FDFMTTRK                    const struct format_descr *
0x00000008  FDFMTEND                    void
0x0000000A  FDSETEMSGTRESH              int
0x0000000B  FDFLUSH                     void
0x0000000C  FDSETMAXERRS                const struct floppy_max_errors *
0x0000000E  FDGETMAXERRS                struct floppy_max_errors *
0x00000010  FDGETDRVTYP                 struct { char [16]; } *
0x00000014  FDSETDRVPRM                 const struct floppy_drive_params *
0x00000015  FDGETDRVPRM                 struct floppy_drive_params *
0x00000016  FDGETDRVSTAT                struct floppy_drive_struct *
0x00000017  FDPOLLDRVSTAT               struct floppy_drive_struct *
0x00000018  FDRESET                     int
0x00000019  FDGETFDCSTAT                struct floppy_fdc_state *
0x0000001B  FDWERRORCLR                 void
0x0000001C  FDWERRORGET                 struct floppy_write_errors *
0x0000001E  FDRAWCMD                    struct floppy_raw_cmd * // MORE // I-O
0x00000028  FDTWADDLE                   void

// <include/linux/fs.h>
0x0000125D  BLKROSET                    const int *
0x0000125E  BLKROGET                    int *
0x0000125F  BLKRRPART                   void
0x00001260  BLKGETSIZE                  int *
0x00001261  BLKFLSBUF                   void
0x00001262  BLKRASET                    int
```


```
0x00001263  BLKRAGET                    int *
0x00000001  FIBMAP                      int *                                  //
0x00000002  FIGETBSZ                    int *

0x00000325  HDIO_SET_NOWERR             int
0x00000326  HDIO_SET_DMA                int

// <include/linux/if_eql.h>
0x000089F0  EQL_ENSLAVE                 struct ifreq *          // MORE // I-O
0x000089F1  EQL_EMANCIPATE              struct ifreq *          // MORE // I-O
0x000089F2  EQL_GETSLAVECFG             struct ifreq *          // MORE // I-O
0x000089F3  EQL_SETSLAVECFG             struct ifreq *          // MORE // I-O
0x000089F4  EQL_GETMASTRCFG             struct ifreq *          // MORE // I-O
0x000089F5  EQL_SETMASTRCFG             struct ifreq *          // MORE // I-O

// <include/linux/if_plip.h>
0x000089F0  SIOCDEVPLIP                 struct ifreq *                 // I-O

// <include/linux/if_ppp.h>
0x00005490  PPPIOCGFLAGS                int *
0x00005491  PPPIOCSFLAGS                const int *
0x00005492  PPPIOCGASYNCMAP             int *
0x00005493  PPPIOCSASYNCMAP             const int *
0x00005494  PPPIOCGUNIT                 int *
0x00005495  PPPIOCSINPSIG               const int *
0x00005497  PPPIOCSDEBUG                const int *
0x00005498  PPPIOCGDEBUG                int *
0x00005499  PPPIOCGSTAT                 struct ppp_stats *
0x0000549A  PPPIOCGTIME                 struct ppp_ddinfo *
0x0000549B  PPPIOCGXASYNCMAP            struct { int [8]; } *
0x0000549C  PPPIOCSXASYNCMAP            const struct { int [8]; } *
0x0000549D  PPPIOCSMRU                  const int *
0x0000549E  PPPIOCRASYNCMAP             const int *
0x0000549F  PPPIOCSMAXCID               const int *

// <include/linux/ipx.h>
0x000089E0  SIOCAIPXITFCRT              const char *
0x000089E1  SIOCAIPXPRISLT              const char *
0x000089E2  SIOCIPXCFGDATA              struct ipx_config_data *

// <include/linux/kd.h>
0x00004B60  GIO_FONT                    struct { char [8192]; } *
0x00004B61  PIO_FONT                    const struct { char [8192]; } *
0x00004B6B  GIO_FONTX                   struct console_font_desc *  // MORE I-O
0x00004B6C  PIO_FONTX                   const struct console_font_desc * //MORE
0x00004B70  GIO_CMAP                    struct { char [48]; } *
0x00004B71  PIO_CMAP                    const struct { char [48]; }
0x00004B2F  KIOCSOUND                   int
0x00004B30  KDMKTONE                    int
0x00004B31  KDGETLED                    char *
0x00004B32  KDSETLED                    int
```

```
0x00004B33  KDGKBTYPE                   char *
0x00004B34  KDADDIO                     int                                 // I-O
0x00004B35  KDDELIO                     int                                 // MORE
0x00004B36  KDENABIO                    void                                // MORE
0x00004B37  KDDISABIO                   void                                // MORE
0x00004B63  KDSKBMETA                   int
0x00004B64  KDGKBLED                    int *
0x00004B65  KDSKBLED                    int
0x00004B46  KDGKBENT                    struct kbentry *                    // I-O
0x00004B47  KDSKBENT                    const struct kbentry *
0x00004B48  KDGKBSENT                   struct kbsentry *                   // I-O
0x00004B49  KDSKBSENT                   const struct kbsentry *
0x00004B4A  KDGKBDIACR                  struct kbdiacrs *
0x00004B4B  KDSKBDIACR                  const struct kbdiacrs *
0x00004B4C  KDGETKEYCODE                struct kbkeycode *                  // I-O
0x00004B4D  KDSETKEYCODE                const struct kbkeycode *
0x00004B4E  KDSIGACCEPT                 int

// <include/linux/lp.h>
0x00000601  LPCHAR                      int
0x00000602  LPTIME                      int
0x00000604  LPABORT                     int
0x00000605  LPSETIRQ                    int
0x00000606  LPGETIRQ                    int *
0x00000608  LPWAIT                      int
0x00000609  LPCAREFUL                   int
0x0000060A  LPABORTOPEN                 int
0x0000060B  LPGETSTATUS                 int *
0x0000060C  LPRESET                     void
0x0000060D  LPGETSTATS                  struct lp_stats *

// <include/linux/mroute.h>
0x000089E0  SIOCGETVIFCNT               struct sioc_vif_req *               // I-O
0x000089E1  SIOCGETSGCNT                struct sioc_sg_req *                // I-O

// <include/linux/mtio.h>
0x40086D01  MTIOCTOP                    const struct mtop *
0x801C6D02  MTIOCGET                    struct mtget *
0x80046D03  MTIOCPOS                    struct mtpos *
0x80206D04  MTIOCGETCONFIG              struct mtconfiginfo *
0x40206D05  MTIOCSETCONFIG              const struct mtconfiginfo *

// <include/linux/netrom.h>
0x000089E0  SIOCNRGETPARMS              struct nr_parms_struct *            // I-O
0x000089E1  SIOCNRSETPARMS              const struct nr_parms_struct *
0x000089E2  SIOCNRDECOBS                void
0x000089E3  SIOCNRRTCTL                 const int *

// <include/linux/sbpcd.h>
0x00009000  DDIOCSDBG                   const int *
0x00005382  CDROMAUDIOBUFSIZ            int
```


```
// <include/linux/scc.h>
0x00005470  TIOCSCCINI                  void
0x00005471  TIOCCHANINI                 const struct scc_modem *
0x00005472  TIOCGKISS                   struct ioctl_command *          // I-O
0x00005473  TIOCSKISS                   const struct ioctl_command *
0x00008910  SIOCGIFNAME                 char []
0x00008911  SIOCSIFLINK                 void
0x00008912  SIOCGIFCONF                 struct ifconf *         // MORE // I-O
0x00008913  SIOCGIFFLAGS                struct ifreq *                  // I-O
0x00008914  SIOCSIFFLAGS                const struct ifreq *
0x00008915  SIOCGIFADDR                 struct ifreq *                  // I-O
0x00008916  SIOCSIFADDR                 const struct ifreq *
0x00008917  SIOCGIFDSTADDR              struct ifreq *                  // I-O
0x00008918  SIOCSIFDSTADDR              const struct ifreq *
0x00008919  SIOCGIFBRDADDR              struct ifreq *                  // I-O
0x0000891A  SIOCSIFBRDADDR              const struct ifreq *
0x0000891B  SIOCGIFNETMASK              struct ifreq *                  // I-O
0x0000891C  SIOCSIFNETMASK              const struct ifreq *
0x0000891D  SIOCGIFMETRIC               struct ifreq *                  // I-O
0x0000891E  SIOCSIFMETRIC               const struct ifreq *
0x0000891F  SIOCGIFMEM                  struct ifreq *                  // I-O
0x00008920  SIOCSIFMEM                  const struct ifreq *
0x00008921  SIOCGIFMTU                  struct ifreq *                  // I-O
0x00008922  SIOCSIFMTU                  const struct ifreq *
0x00008923  OLD_SIOCGIFHWADDR           struct ifreq *                  // I-O
0x00008924  SIOCSIFHWADDR               const struct ifreq *            // MORE
0x00008925  SIOCGIFENCAP                int *
0x00008926  SIOCSIFENCAP                const int *
0x00008927  SIOCGIFHWADDR               struct ifreq *                  // I-O
0x00008929  SIOCGIFSLAVE                void
0x00008930  SIOCSIFSLAVE                void
0x00008931  SIOCADDMULTI                const struct ifreq *
0x00008932  SIOCDELMULTI                const struct ifreq *
0x00008940  SIOCADDRTOLD                void
0x00008941  SIOCDELRTOLD                void
0x00008950  SIOCDARP                    const struct arpreq *
0x00008951  SIOCGARP                    struct arpreq *                 // I-O
0x00008952  SIOCSARP                    const struct arpreq *
0x00008960  SIOCDRARP                   const struct arpreq *
0x00008961  SIOCGRARP                   struct arpreq *                 // I-O
0x00008962  SIOCSRARP                   const struct arpreq *
0x00008970  SIOCGIFMAP                  struct ifreq *                  // I-O
0x00008971  SIOCSIFMAP                  const struct ifreq *

// <include/linux/soundcard.h>
0x00005100  SNDCTL_SEQ_RESET            void
0x00005101  SNDCTL_SEQ_SYNC             void
0xC08C5102  SNDCTL_SYNTH_INFO           struct synth_info *             // I-O
0xC0045103  SNDCTL_SEQ_CTRLRATE         int *                           // I-O
0x80045104  SNDCTL_SEQ_GETOUTCOUNT      int *
```


```
0x80045105  SNDCTL_SEQ_GETINCOUNT       int *
0x40045106  SNDCTL_SEQ_PERCMODE         void
0x40285107  SNDCTL_FM_LOAD_INSTR        const struct sbi_instrument *
0x40045108  SNDCTL_SEQ_TESTMIDI         const int *
0x40045109  SNDCTL_SEQ_RESETSAMPLES     const int *
0x8004510A  SNDCTL_SEQ_NRSYNTHS         int *
0x8004510B  SNDCTL_SEQ_NRMIDIS          int *
0x40045408  SNDCTL_TMR_SELECT           int *                           // I-O
0xCFB85001  SNDCTL_PMGR_IFACE           struct patmgr_info *            // I-O
0xC0046D00  SNDCTL_MIDI_PRETIME         int *                           // I-O
0xC0046D01  SNDCTL_MIDI_MPUMODE         const int *
0xC0216D02  SNDCTL_MIDI_MPUCMD          struct mpu_command_rec *        // I-O
0x00005000  SNDCTL_DSP_RESET            void
0x00005001  SNDCTL_DSP_SYNC             void
0xC0045002  SNDCTL_DSP_SPEED            int *                           // I-O
0xC0045003  SNDCTL_DSP_STEREO           int *                           // I-O
0xC0045004  SNDCTL_DSP_GETBLKSIZE       int *                           // I-O
0xC0045006  SOUND_PCM_WRITE_CHANNELS    int *                           // I-O
0xC0045007  SOUND_PCM_WRITE_FILTER      int *                           // I-O
0x00005008  SNDCTL_DSP_POST             void
0xC0045009  SNDCTL_DSP_SUBDIVIDE        int *                           // I-O
0xC004500A  SNDCTL_DSP_SETFRAGMENT      int *                           // I-O
0x8004500B  SNDCTL_DSP_GETFMTS          int *
0xC0045005  SNDCTL_DSP_SETFMT           int *                           // I-O
0x800C500C  SNDCTL_DSP_GETOSPACE        struct audio_buf_info *
0x800C500D  SNDCTL_DSP_GETISPACE        struct audio_buf_info *
0x0000500E  SNDCTL_DSP_NONBLOCK         void
0x80045002  SOUND_PCM_READ_RATE         int *
0x80045006  SOUND_PCM_READ_CHANNELS     int *
0x80045005  SOUND_PCM_READ_BITS         int *
0x80045007  SOUND_PCM_READ_FILTER       int *
0x00004300  SNDCTL_COPR_RESET           void
0xCFB04301  SNDCTL_COPR_LOAD            const struct copr_buffer *
0xC0144302  SNDCTL_COPR_RDATA           struct copr_debug_buf *         // I-O
0xC0144303  SNDCTL_COPR_RCODE           struct copr_debug_buf *         // I-O
0x40144304  SNDCTL_COPR_WDATA           const struct copr_debug_buf *
0x40144305  SNDCTL_COPR_WCODE           const struct copr_debug_buf *
0xC0144306  SNDCTL_COPR_RUN             struct copr_debug_buf *         // I-O
0xC0144307  SNDCTL_COPR_HALT            struct copr_debug_buf *         // I-O
0x4FA44308  SNDCTL_COPR_SENDMSG         const struct copr_msg *
0x8FA44309  SNDCTL_COPR_RCVMSG          struct copr_msg *
0x80044D00  SOUND_MIXER_READ_VOLUME     int *
0x80044D01  SOUND_MIXER_READ_BASS       int *
0x80044D02  SOUND_MIXER_READ_TREBLE     int *
0x80044D03  SOUND_MIXER_READ_SYNTH      int *
0x80044D04  SOUND_MIXER_READ_PCM        int *
0x80044D05  SOUND_MIXER_READ_SPEAKER    int *
0x80044D06  SOUND_MIXER_READ_LINE       int *
0x80044D07  SOUND_MIXER_READ_MIC        int *
0x80044D08  SOUND_MIXER_READ_CD         int *
0x80044D09  SOUND_MIXER_READ_IMIX       int *
```


```
0x80044D0A  SOUND_MIXER_READ_ALTPCM     int *
0x80044D0B  SOUND_MIXER_READ_RECLEV     int *
0x80044D0C  SOUND_MIXER_READ_IGAIN      int *
0x80044D0D  SOUND_MIXER_READ_OGAIN      int *
0x80044D0E  SOUND_MIXER_READ_LINE1      int *
0x80044D0F  SOUND_MIXER_READ_LINE2      int *
0x80044D10  SOUND_MIXER_READ_LINE3      int *
0x80044D1C  SOUND_MIXER_READ_MUTE       int *
0xC0044D07  SOUND_MIXER_WRITE_MIC       int *                           // I-O
0xC0044D08  SOUND_MIXER_WRITE_CD        int *                           // I-O
0xC0044D09  SOUND_MIXER_WRITE_IMIX      int *                           // I-O
0xC0044D0A  SOUND_MIXER_WRITE_ALTPCM    int *                           // I-O
0xC0044D0B  SOUND_MIXER_WRITE_RECLEV    int *                           // I-O
0xC0044D0C  SOUND_MIXER_WRITE_IGAIN     int *                           // I-O
0xC0044D0D  SOUND_MIXER_WRITE_OGAIN     int *                           // I-O
0xC0044D0E  SOUND_MIXER_WRITE_LINE1     int *                           // I-O
0xC0044D0F  SOUND_MIXER_WRITE_LINE2     int *                           // I-O
0xC0044D10  SOUND_MIXER_WRITE_LINE3     int *                           // I-O
0xC0044D1C  SOUND_MIXER_WRITE_MUTE      int *                           // I-O
0xC0044D1D  SOUND_MIXER_WRITE_ENHANCE   int *                           // I-O
0xC0044D1E  SOUND_MIXER_WRITE_LOUD      int *                           // I-O
0xC0044DFF  SOUND_MIXER_WRITE_RECSRC    int *                           // I-O

// <include/linux/umsdos_fs.h>
0x000004D2  UMSDOS_READDIR_DOS          struct umsdos_ioctl *           // I-O
0x000004D3  UMSDOS_UNLINK_DOS           const struct umsdos_ioctl *
0x000004D4  UMSDOS_RMDIR_DOS            const struct umsdos_ioctl *
0x000004D5  UMSDOS_STAT_DOS             struct umsdos_ioctl *           // I-O
0x000004D6  UMSDOS_CREAT_EMD            const struct umsdos_ioctl *
0x000004D7  UMSDOS_UNLINK_EMD           const struct umsdos_ioctl *
0x000004D8  UMSDOS_READDIR_EMD          struct umsdos_ioctl *           // I-O
0x000004D9  UMSDOS_GETVERSION           struct umsdos_ioctl *
0x000004DA  UMSDOS_INIT_EMD             void
0x000004DB  UMSDOS_DOS_SETUP            const struct umsdos_ioctl *
0x000004DC  UMSDOS_RENAME_DOS           const struct umsdos_ioctl *

// <include/linux/vt.h>
0x00005600  VT_OPENQRY                  int *
0x00005601  VT_GETMODE                  struct vt_mode *
0x00005602  VT_SETMODE                  const struct vt_mode *
0x00005603  VT_GETSTATE                 struct vt_stat *
0x00005604  VT_SENDSIG                  void
0x00005605  VT_RELDISP                  int
0x00005606  VT_ACTIVATE                 int
0x00005607  VT_WAITACTIVE               int
0x00005608  VT_DISALLOCATE              int
0x00005609  VT_RESIZE                   const struct vt_sizes *
0x0000560A  VT_RESIZEX                  const struct vt_consize *
```

// More arguments.

Some ioctl's take a pointer to a structure which contains additional pointers. These are documented here in alphabetical order.

CDROMREADAUDIO takes an input pointer 'const struct cdrom_read_audio *'. The 'buf' field points to an output buffer of length

CDROMREADCOOKED, CDROMREADMODE1, CDROMREADMODE2, and CDROMREADRAW take

```
            EQL_ENSLAVE                 const struct slaving_request *
            EQL_EMANCIPATE              const struct slaving_request *
            EQL_GETSLAVECFG             struct slave_config *           // I-O
            EQL_SETSLAVECFG             const struct slave_config *
            EQL_GETMASTERCFG            struct master_config *
            EQL_SETMASTERCFG            const struct master_config *
```

FDRAWCMD takes a 'struct floppy raw_cmd *'. If 'flags & FD_RAW_WRITE' is non-zero, then 'data' points to an input buffer of length 'length'. If 'flags & FD_RAW_READ' is non-zero, then 'data' points to an output buffer of length 'length'.

GIO_FONTX and PIO_FONTX take a 'struct console_font_desc *' or a a buffer of 'char [charcount]'. This is an output buffer for GIO_FONTX and an input buffer for PIO_FONTX.

GIO_UNIMAP and PIO_UNIMAP take a 'struct unimapdesc *' or a of 'struct unipair [entry_ct]'. This is an output buffer for GIO_UNIMAP and an input buffer for PIO_UNIMAP.

KDADDIO, KDDELIO, KDDISABIO, and KDENABIO enable or disable access to I/O ports. They are essentially alternate interfaces to 'ioperm'.

KDMAPDISP and KDUNMAPDISP enable or disable memory mappings or I/O port access. They are not implemented in the kernel.

SCSI_IOCTL_PROBE_HOST takes an input pointer 'const int *', which is a length. It uses the same pointer as an output pointer to a 'char []' buffer of this length.

SIOCADDRT and SIOCDELRT take an input pointer whose type depends on the protocol:

```
            Most protocols              const struct rtentry *
            AX.25                       const struct ax25_route *
            NET/ROM                     const struct nr_route_struct *
```

SIOCGIFCONF takes a 'struct ifconf *'. The 'ifc_buf' field points to a buffer of length 'ifc_len' bytes, into which the kernel writes a list of type 'struct ifreq []'.

SIOCSIFHWADDR takes an input pointer whose type depends on the protocol:

Most protocols	const struct ifreq *
AX.25	const char [AX25_ADDR_LEN]

TIOCLINUX takes a 'const char *'. It uses this to distinguish several independent sub-cases. In the table below, 'N + foo' means 'foo' after an N-byte pad. 'struct selection' is implicitly defined in

TIOCLINUX-2	1 + const struct selection *

SIOCPROTOPRIVATE.

0x00000001	FDSETPRM	FIBMAP
0x00000002	FDDEFPRM	FIGETBSZ
0x00005382	CDROMAUDIOBUFSIZ	SCSI_IOCTL_GET_IDLUN
0x00005402	SNDCTL_TMR_START	TCSETS
0x00005403	SNDCTL_TMR_STOP	TCSETSW
0x00005404	SNDCTL_TMR_CONTINUE	TCSETSF

NAME

ioperm - set port input/output permissions

SYNOPSIS

#include <unistd.h>

int ioperm(unsigned long *from*, **unsigned long** *num*, **int** *turn_on*

DESCRIPTION

Ioperm sets the port access permission bits for the process for *num* bytes starting from port address **from** to the value **turn_on**. The use of **ioperm** requires root privileges.

Only the first 0x3ff I/O ports can be specified in this manner. For more ports, the **iopl** function must be used. Permissions are not inherited on fork, but on exec they are. This is useful for giving port access permissions to non-privileged tasks.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

CONFORMING TO

ioperm is Linux specific and should not be used in programs intended to be portable.

SEE ALSO

iopl(2)

TOC

NAME

iopl - change I/O privilege level

SYNOPSIS

#include <unistd.h>

int iopl(int *level*));

DESCRIPTION

iopl changes the I/O privilege level of the current process, as specified in *level*.

This call is necessary to allow 8514-compatible X servers to run under Linux. Since these X servers require access to all 65536 I/O ports, the **ioperm** call is not sufficient.

In addition to granting unrestricted I/O port access, running at a higher I/O privilege level also allows the process to disable interrupts. This will probably crash the system, and is not recommended.

Permissions are inherited by fork and exec.

The I/O privilege level for a normal process is 0.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL *level* is greater than 3.

EPERM The current user is not the super-user.

GLIBC NOTE

Under libc5, the prototype for *iopl*() is given in <*unistd.h*>, but glibc2 has this prototype in <*sys/io.h*>.

NOTES FROM THE KERNEL SOURCE

iopl has to be used when you want to access the I/O ports beyond the 0x3ff range: to get the full 65536 ports bit-mapped you'd need 8kB of bitmaps/process, which is a bit excessive.

CONFORMING TO

iopl is Linux specific and should not be used in processes intended to be portable.

SEE ALSO

ioperm(2)

NAME

ipc - System V IPC system calls

SYNOPSIS

int ipc(unsigned int *call*, **int** *first*, **int** *second*, **int** *third*, **void** **ptr*, **long** *fifth*);

DESCRIPTION

ipc is a common kernel entry point for the System V IPC calls for messages, semaphores, and shared memory. *call* determines which IPC function to invoke; the other arguments are passed through to the appropriate call.

User programs should call the appropriate functions by their usual names. Only standard library implementors and kernel hackers need to know about **ipc**.

CONFORMING TO

ipc is Linux specific, and should not be used in programs intended to be portable.

SEE ALSO

msgctl(2), **msgget**(2), **msgrcv**(2), **msgsnd**(2), **semctl**(2), **semget**(2), **semop**(2), **shmat**(2), **shmctl**(2), **shmdt**(2), **shmget**(2)

NAME

kill - send signal to a process

SYNOPSIS

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int sig));
```

DESCRIPTION

The **kill** system call can be used to send any signal to any process group or process.

If *pid* is positive, then signal *sig* is sent to *pid*.

If *pid* equals 0, then *sig* is sent to every process in the process group of the current process.

If *pid* equals -1, then *sig* is sent to every process except for the first one, from higher numbers in the process table to lower.

If *pid* is less than -1, then *sig* is sent to every process in the process group -*pid*.

If *sig* is 0, then no signal is sent, but error checking is still performed.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
An invalid signal was specified.

ESRCH
The pid or process group does not exist. Note that an existing process might be a zombie, a process which already committed termination, but has not yet been **wait()**ed for.

EPERM
The process does not have permission to send the signal to any of the receiving processes. For a process to have permission to send a signal to process *pid* it must either have root privileges, or the real or effective user ID of the sending process must equal the real or saved set-user-ID of the receiving process.

BUGS

It is impossible to send a signal to task number one, the init process, for which it has not installed a signal handler. This is done to assure the system is not brought down accidentally.

CONFORMING TO

SVr4, SVID, POSIX.1, X/OPEN, BSD 4.3

SEE ALSO

_exit(2), **exit**(3), **signal**(2),

TOC

NAME

killpg - send signal to a process group

SYNOPSIS

```
#include <signal.h>

int killpg(int pgrp, int sig));
```

DESCRIPTION

Killpg sends the signal *sig* to the process group *pgrp*. See **sigaction**(2) for a list of signals. If *pgrp* is 0, **killpg** sends the signal to the sending process's process group.

The sending process and members of the process group must have the same effective user ID, or the sender must be the super-user. As a single special case the continue signal **SIGCONT** may be sent to any process that is a descendant of the current process.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
Sig is not a valid signal number.

ESRCH
No process can be found in the process group specified by *pgrp*.

ESRCH
The process group was given as 0 but the sending process does not have a process group.

EPERM
The sending process is not the super-user and one or more of the target processes has an effective user ID different from that of the sending process.

CONFORMING TO

SVr4, 4.4BSD (The **killpg** function call first appeared in 4.0BSD).

SEE ALSO

kill(2), **getpgrp**(2), **signal**(2)

NAME

link - make a new name for a file

SYNOPSIS

```
#include <unistd.h>

int link(const char *oldpath, const char *newpath));
```

DESCRIPTION

link creates a new link (also known as a hard link) to an existing file.

If *newpath* exists it will *not* be overwritten.

This new name may be used exactly as the old one for any operation; both names refer to the same file (and so have the same permissions and ownership) and it is impossible to tell which name was the `original'.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EXDEV *oldpath* and *newpath* are not on the same filesystem.

EPERM The filesystem containing *oldpath* and *newpath* does not support the creation of hard links.

EFAULT *oldpath* or *newpath* points outside your accessible address space.

EACCES Write access to the directory containing *newpath* is not allowed for the process's effective uid, or one of the directories in *oldpath* or *newpath* did not allow search (execute) permission.

ENAMETOOLONG
oldpath or *newpath* was too long.

ENOENT A directory component in *oldpath* or *newpath* does not exist or is a dangling symbolic link.

ENOTDIR A component used as a directory in *oldpath* or *newpath* is not, in fact, a directory.

ENOMEM Insufficient kernel memory was available.

EROFS The file is on a read-only filesystem.

EEXIST *newpath* already exists.

EMLINK The file referred to by *oldpath* already has the maximum number of links to it.

ELOOP Too many symbolic links were encountered in resolving *oldpath* or *newpath*.

ENOSPC The device containing the file has no room for the new directory entry.

EPERM *oldpath* is a directory.

EIO An I/O error occurred.

NOTES

Hard links, as created by **link**, cannot span filesystems. Use **symlink** if this is required.

CONFORMING TO

SVr4, SVID, POSIX, BSD 4.3, X/OPEN. SVr4 documents additional ENOLINK and EMULTIHOP error conditions; POSIX.1 does not document ELOOP. X/OPEN does not document EFAULT, ENOMEM or EIO.

BUGS

On NFS file systems, the return code may be wrong in case the NFS server performs the link creation and dies before it can say so. Use *stat*(2) to find out if the link got created.

SEE ALSO

symlink(2), **unlink**(2), **rename**(2), **open**(2), **stat**(2), **ln**(1)

TOC

TOC

NAME

listen - listen for connections on a socket

SYNOPSIS

#include <sys/socket.h>

int listen(int s, int *backlog*));

DESCRIPTION

To accept connections, a socket is first created with **socket**(2), a willingness to accept incoming connections and a queue limit for incoming connections are specified with **listen**, and then the connections are accepted with **accept**(2). The **listen** call applies only to sockets of type **SOCK_STREAM** or **SOCK_SEQPACKET**.

The *backlog* parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full the client may receive an error with an indication of **ECONNREFUSED**, or, if the underlying protocol supports retransmission, the request may be ignored so that retries may succeed.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF The argument **s** is not a valid descriptor.

ENOTSOCK
The argument **s** is not a socket.

EOPNOTSUPP
The socket is not of a type that supports the operation **listen**.

CONFORMING TO

SVr4, 4.4BSD (the **listen** function call first appeared in 4.2BSD).

BUGS

If the socket is of type af_inet, and the *backlog* argument is greater than the constant SO_MAXCONN (128 in 2.0.23), it is silently truncated to SO_MAXCONN. For portable applications don't rely on this value since BSD (and at least some BSD derived systems) limit the backlog to 5.

SEE ALSO

accept(2), **connect**(2), **socket**(2)

NAME

_llseek - reposition read/write file offset

SYNOPSIS

#include <unistd.h>

#include <linux/unistd.h>

_syscall5(int, _llseek, uint, fd, ulong, hi,

int _llseek(unsigned int *fd*, **unsigned long** *offset_high*, **unsigned long** *offset_low*

DESCRIPTION

The **_llseek** function repositions the offset of the file descriptor *fd* to (*offset_high<<32*) | *offset_low* bytes relative to the beginning of the file, the current position in the file, or the end of the file, depending on whether *whence* is **SEEK_SET**, **SEEK_CUR**, or **SEEK_END**, respectively. It returns the resulting file position in the argument *result*.

RETURN VALUES

Upon successful completion, **_llseek** returns 0. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

ERRORS

EBADF
fd is not an open file descriptor.

EINVAL
whence is invalid.

CONFORMING TO

This function is Linux-specific, and should not be used in programs intended to be portable.

BUGS

The ext2 filesystem does not support files with a size of 2GB or more.

SEE ALSO

lseek(2)

NAME

lseek - reposition read/write file offset

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence
```

DESCRIPTION

The **lseek** function repositions the offset of the file descriptor *fildes* to the argument *offset* according to the directive *whence* as follows:

SEEK_SET
The offset is set to *offset* bytes.

SEEK_CUR
The offset is set to its current location plus *offset* bytes.

SEEK_END
The offset is set to the size of the file plus *offset* bytes.

The **lseek** function allows the file offset to be set beyond the end of the existing end-of-file of the file. If data is later written at this point, subsequent reads of the data in the gap return bytes of zeros (until data is actually written into the gap).

RETURN VALUES

Upon successful completion, **lseek** returns the resulting offset location as measured in bytes from the beginning of the file. Otherwise, a value of (off_t)-1 is returned and *errno* is set to indicate the error.

ERRORS

EBADF
Fildes is not an open file descriptor.

ESPIPE
Fildes is associated with a pipe, socket, or FIFO.

EINVAL
Whence is not a proper value.

CONFORMING TO

SVr4, POSIX, BSD 4.3

RESTRICTIONS

Some devices are incapable of seeking and POSIX does not specify which devices must support it.

Linux specific restrictions: using **lseek** on a tty device returns **ESPIPE**. Other systems return the number of written characters, using SEEK_SET to set the counter. Some devices, e.g. **/dev/null** do not cause the error **ESPIPE**, but

return a pointer which value is undefined.

NOTES

This document's use of *whence* is incorrect English, but maintained for historical reasons.

When converting old code, substitute values for *whence* with the following macros:

c c l l. old new 0 SEEK_SET 1 SEEK_CUR 2 SEEK_END
L_SET SEEK_SET L_INCR SEEK_CUR L_XTND SEEK_END

SVR1-3 returns **long** instead of **off_t**, BSD returns **int**.

SEE ALSO

dup(2), **open**(2), **fseek**(3)

TOC

NAME

mkdir - create a directory

SYNOPSIS

```
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>

int mkdir(const char *pathname, mode_t mode));
```

DESCRIPTION

mkdir attempts to create a directory named *pathname*.

mode specifies the permissions to use. It is modified by the process's **umask** in the usual way: the permissions of the created file are **(mode & ~umask)**.

The newly created directory will be owned by the effective uid of the process. If the directory containing the file has the set group id bit set, or if the filesystem is mounted with BSD group semantics, the new directory will inherit the group ownership from its parent; otherwise it will be owned by the effective gid of the process.

If the parent directory has the set group id bit set then so will the newly created directory.

RETURN VALUE

mkdir returns zero on success, or -1 if an error occurred (in which case, *errno* is set appropriately).

ERRORS

EEXIST *pathname* already exists (not necessarily as a directory).

EFAULT *pathname* points outside your accessible address space.

EACCES The parent directory does not allow write permission to the process, or one of the directories in *pathname* did not allow search (execute) permission.

ENAMETOOLONG
pathname was too long.

ENOENT A directory component in *pathname* does not exist or is a dangling symbolic link.

ENOTDIR A component used as a directory in *pathname* is not, in fact, a directory.

ENOMEM Insufficient kernel memory was available.

EROFS *pathname* refers to a file on a read-only filesystem.

ELOOP Too many symbolic links were encountered in resolving *pathname*.

ENOSPC The device containing *pathname* has no room for the new directory.

ENOSPC The new directory cannot be created because the user's disk quota is exhausted.

CONFORMING TO

SVr4, POSIX, BSD, SYSV, X/OPEN. SVr4 documents additional EIO, EMULTIHOP and ENOLINK error conditions; POSIX.1 omits ELOOP.

There are many infelicities in the protocol underlying NFS. Some of these affect **mkdir**.

SEE ALSO

read(2), **write**(2), **fcntl**(2), **unlink**(2), **open**(2), **mknod**(2), **mount**(2), **socket**(2), **socket**(2),

TOC

NAME

mknod - create a directory or special or ordinary file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int mknod(const char *pathname, mode_t mode, dev_t dev
```

DESCRIPTION

mknod attempts to create a filesystem node (file, device special file or named pipe) named *pathname*, specified by *mode* and *dev*.

mode specifies both the permissions to use and the type of node to be created.

It should be a combination (using bitwise OR) of one of the file types listed below and the permissions for the new node.

The permissions are modified by the process's **umask** in the usual way: the permissions of the created node are **(mode & ~umask)**.

The file type should be one of **S_IFREG**, **S_IFCHR**, **S_IFBLK** and to specify a normal file (which will be created empty), character special file, block special file or FIFO (named pipe), respectively, or zero, which will create a normal

file.

If the file type is **S_IFCHR** or **S_IFBLK** then *dev* specifies the major and minor numbers of the newly created device special file; otherwise it is ignored.

The newly created node will be owned by the effective uid of the process. If the directory containing the node has the set group id bit set, or if the filesystem is mounted with BSD group semantics, the new node will inherit the group ownership from its parent directory; otherwise it will be owned by the effective gid of the process.

RETURN VALUE

mknod returns zero on success, or -1 if an error occurred (in which case, *errno* is set appropriately).

ERRORS

EPERM
mode requested creation of something other than a FIFO (named pipe), and the caller is not the superuser; also returned if the filesystem containing *pathname* does not support the type of node requested.

EINVAL
mode requested creation of something other than a normal file, device special file or FIFO.

EEXIST
pathname already exists.

EFAULT
pathname points outside your accessible address space.

EACCES
The parent directory does not allow write permission to

the process, or one of the directories in *pathname* did not allow search (execute) permission.

ENAMETOOLONG
pathname was too long.

ENOENT
A directory component in *pathname* does not exist or is a dangling symbolic link.

ENOTDIR
A component used as a directory in *pathname* is not, in fact, a directory.

ENOMEM
Insufficient kernel memory was available.

EROFS
pathname refers to a file on a read-only filesystem.

ELOOP
Too many symbolic links were encountered in resolving *pathname*.

ENOSPC
The device containing *pathname* has no room for the new node.

CONFORMING TO

SVr4 (but the call requires privilege and is thus not in POSIX), 4.4BSD. The Linux version differs from the SVr4 version in that it does not require root permission to create pipes, also in that no EMULTIHOP, ENOLINK, or EINTR error is documented.

BUGS

The **mknod** call cannot be used to create directories or socket files, and cannot be used to create normal files by users other than the superuser.

There are many infelicities in the protocol underlying NFS. Some of these affect **mknod**.

SEE ALSO

read(2), **write**(2), **fcntl**(2), **unlink**(2), **open**(2), **mkdir**(2), **mount**(2), **socket**(2), **fopen**(3).

TOC

TOC

NAME

mlock - disable paging for some parts of memory

SYNOPSIS

#include <sys/mman.h>

int mlock(const void * *addr*, **size_t** *len*));

DESCRIPTION

mlock disables paging for the memory in the range starting at *addr* with length *len* bytes. All pages which contain a part of the specified memory range are guaranteed be resident in RAM when the **mlock** system call returns successfully and they are guaranteed to stay in RAM until the pages are unlocked by **munlock** or **munlockall**, or until the process terminates or starts another program with **exec**. Child processes do not inherit page locks across a **fork**.

Memory locking has two main applications: real-time algorithms and high-security data processing. Real-time applications require deterministic timing, and, like scheduling, paging is one major cause of unexpected program execution delays. Real-time applications will usually also switch to a real-time scheduler with **sched_setscheduler**. Cryptographic security software often handles critical bytes like passwords or secret keys as data structures. As a result of paging, these secrets could be transfered onto a persistent swap store medium, where they might be accessible to the enemy long after the security software has erased the secrets in RAM and terminated.


TOC

Memory locks do not stack, i.e., pages which have been locked several times by calls to **mlock** or **mlockall** will be unlocked by a single call to **munlock** for the corresponding range or by **munlockall**. Pages which are mapped to several locations or by several processes stay locked into RAM as long as they are locked at least at one location or by at least one process.

On POSIX systems on which **mlock** and **munlock** are available, **_POSIX_MEMLOCK_RANGE** is defined in <unistd.h> and the value **PAGESIZE** from <limits.h> indicates the number of bytes per page.

RETURN VALUE

On success, **mlock** returns zero. On error, -1 is returned, *errno* is set appropriately, and no changes are made to any locks in the address space of the process.

ERRORS

ENOMEM Some of the specified address range does not correspond to mapped pages in the address space of the process or the process tried to exceed the maximum number of allowed locked pages.

EPERM The calling process does not have appropriate privileges. Only root processes are allowed to lock pages.

EINVAL *len* was not a positive number.

CONFORMING TO

POSIX.1b, SVr4. SVr4 documents an additional EAGAIN error code.

SEE ALSO

munlock(2), **mlockall**(2), and **munlockall**(2).

TOC

TOC

NAME

mlockall - disable paging for calling process

SYNOPSIS

#include <sys/mman.h>

int mlockall(int *flags)*);

DESCRIPTION

mlockall disables paging for all pages mapped into the address space of the calling process. This includes the pages of the code, data and stack segment, as well as shared libraries, user space kernel data, shared memory and memory mapped files. All mapped pages are guaranteed to be resident in RAM when the **mlockall** system call returns successfully and they are guaranteed to stay in RAM until the pages are unlocked again by **munlock** or **munlockall** or until the process terminates or starts another program with **exec**. Child processes do not inherit page locks across a **fork**.

Memory locking has two main applications: real-time algorithms and high-security data processing. Real-time applications require deterministic timing, and, like scheduling, paging is one major cause of unexpected program execution delays. Real-time applications will usually also switch to a real-time scheduler with **sched_setscheduler**. Cryptographic security software often handles critical bytes like passwords or secret keys as data structures. As a result of paging, these secrets could be transfered onto a persistent swap store medium, where they might be accessible to the enemy long after the security software has erased the

secrets in RAM and terminated. For security applications, only small parts of memory have to be locked, for which **mlock** is available.

The *flags* parameter can be constructed from the logical OR of the following constants:

MCL_CURRENT Lock all pages which are currently mapped into the address space of the process.

MCL_FUTURE Lock all pages which will become mapped into the address space of the process in the future. These could be for instance new pages required by a growing heap and stack as well as new memory mapped files or shared memory regions.

If **MCL_FUTURE** has been specified and the number of locked pages exceeds the upper limit of allowed locked pages, then the system call which caused the new mapping will fail with **ENOMEM**. If these new pages have been mapped by the the growing stack, then the kernel will deny stack expansion and send a **SIGSEGV**.

Real-time processes should reserve enough locked stack pages before entering the time-critical section, so that no page fault can be caused by function calls. This can be achieved by calling a function which has a sufficiently large automatic variable and which writes to the memory occupied by this large array in order to touch these stack pages. This way, enough pages will be mapped for the stack and can be locked into RAM. The dummy writes ensure that not even copy-on-write page faults can occur in the critical section.

Memory locks do not stack, i.e., pages which have been locked several times by calls to **mlockall** or **mlock** will be unlocked by a single call to **munlockall**. Pages which are mapped to several locations or by several processes stay locked into RAM as long as they are locked at least at one location or by at least one process.

On POSIX systems on which **mlockall** and **munlockall** are available, **_POSIX_MEMLOCK** is defined in <unistd.h>.

RETURN VALUE

On success, **mlockall** returns zero. On error, -1 is returned, *errno* is set appropriately.

ERRORS

ENOMEM
The process tried to exceed the maximum number of allowed locked pages.

EPERM
The calling process does not have appropriate privileges. Only root processes are allowed to lock pages.

EINVAL
Unknown flags were specified.

CONFORMING TO

POSIX.1b, SVr4. SVr4 documents an additional EAGAIN error code.

SEE ALSO

munlockall(2), **mlock**(2), and **munlock**(2).

TOC

NAME

mmap, munmap - map or unmap files or devices into memory

SYNOPSIS

```
#include <unistd.h>
#include <sys/mman.h>

#ifdef _POSIX_MAPPED_FILES

void * mmap(void *start, size_t length, int prot , int
flags, int fd, off_t offset

int munmap(void *start, size_t length));

#endif
```

DESCRIPTION

The **mmap** function asks to map *length* bytes starting at offset *offset* from the file (or other object) specified by *fd* into memory, preferably at address *start*. This latter address is a hint only, and is usually specified as 0. The actual place where the object is mapped is returned by **mmap**. The *prot* argument describes the desired memory protection. It has bits

PROT_EXEC Pages may be executed.

PROT_READ Pages may be read.

PROT_WRITE Pages may be written.

TOC

PROT_NONE Pages may not be accessed.

The *flags* parameter specifies the type of the mapped object, mapping options and whether modifications made to the mapped copy of the page are private to the process or are to be shared with other references. It has bits

MAP_FIXED Do not select a different address than the one specified. If the specified address cannot be used, **mmap** will fail. If MAP_FIXED is specified, *start* must be a multiple of the pagesize. Use of this option is discouraged.

MAP_SHARED Share this mapping with all other processes that map this object

MAP_PRIVATE
Create a private copy-on-write mapping.

You must specify exactly one of MAP_SHARED and MAP_PRIVATE.

The above three flags are described in POSIX.1b (formerly POSIX.4). Linux also knows about MAP_DENYWRITE, MAP_EXECUTABLE and MAP_ANON(YMOUS).

The **munmap** system call deletes the mappings for the specified address range, and causes further references to addresses within the range to generate invalid memory references.

RETURN VALUE

On success, **mmap** returns a pointer to the mapped area. On error, MAP_FAILED (-1) is returned, and *errno* is set appropriately. On success, **munmap** returns 0, on failure -1, and *errno* is set (probably to EINVAL).

ERRORS

EBADF
fd is not a valid file descriptor (and MAP_ANONYMOUS was not set).

EACCES
MAP_PRIVATE was asked, but *fd* is not open for reading. Or MAP_SHARED was asked and PROT_WRITE is set, *fd* is not open for writing.

EINVAL
We don't like *start* or *length* or *offset*. (E.g., they are too large, or not aligned on a PAGESIZE boundary.)

ETXTBUSY
MAP_DENYWRITE was set but the object specified by *fd* is open for writing.

EAGAIN
The file has been locked, or too much memory has been locked.

ENOMEM
No memory is available.

CONFORMING TO

SVr4, POSIX.1b (formerly POSIX.4), 4.4BSD. Svr4 documents additional error codes ENXIO and ENODEV.

SEE ALSO

getpagesize(2), **msync**(2), **shm_open**(2), B.O. Gallmeister, POSIX.4, O'Reilly, pp. 128-129 and 389-391.


TOC

TOC

NAME

modify_ldt - get or set ldt

SYNOPSIS

```
#include <linux/ldt.h>
#include <linux/unistd.h>

_syscall3( int, modify_ldt, int, func, void

int modify_ldt(int func, void *ptr, unsigned long bytecount
```

DESCRIPTION

modify_ldt reads or writes the local descriptor table (ldt) for a process. The ldt is a per-process memory management table used by the i386 processor. For more information on this table, see an Intel 386 processor handbook.

When *func* is 0, **modify_ldt** reads the ldt into the memory pointed to by *ptr*. The number of bytes read is the smaller of *bytecount* and the actual size of the ldt.

When *func* is 1, **modify_ldt** modifies one ldt entry. *ptr* points to a *modify_ldt_ldt_s* structure and *bytecount* must equal the size of this structure.

RETURN VALUE

On success, **modify_ldt** returns either the actual number of bytes read (for reading) or 0 (for writing). On failure, **modify_ldt** returns -1 and sets *errno*.

ERRORS

ENOSYS
func is neither 0 nor 1.

EINVAL
ptr is 0, or *func* is 1 and *bytecount* is not equal to the size of the structure *modify_ldt_ldt_s*, or *func* is 1 and the new ldt entry has illegal values.

EFAULT
ptr points outside the address space.

CONFORMING TO

This call in Linux-specfic and should not be used in programs intended to be portable.

SEE ALSO

vm86(2)

NAME

mount, umount - mount and unmount filesystems.

SYNOPSIS

```
#include <sys/mount.h>
#include <linux/fs.h>        /* very unwise */

int mount(const char *specialfile, const char * dir ,  const
char  *  filesystemtype, unsigned long rwflag , const void *
data));

int umount(const char *specialfile));

int umount(const char *dir));
```

DESCRIPTION

mount attaches the filesystem specified by *specialfile* (which is often a device name) to the directory specified by *dir*.

umount removes the attachment of the filesystem specified by *specialfile* or *dir*.

Only the super-user may mount and unmount filesystems.

The *filesystemtype* argument may take one of the values listed in /proc/filesystems (like "minix", "ext2", "msdos", "proc", "nfs", "iso9660" etc.).

The *rwflag* argument has the magic number 0xC0ED in the top 16 bits, and various mount flags (as defined in <linux/fs.h>) in the low order 16 bits:


```
#define MS_RDONLY    1 /* mount read-only */
#define MS_NOSUID    2 /* ignore suid and sgid bits */
#define MS_NODEV     4 /* disallow access to device special files */
#define MS_NOEXEC    8 /* disallow program execution */
#define MS_SYNC     16 /* writes are synced at once */
#define MS_REMOUNT  32 /* alter flags of a mounted FS */
#define MS_MGC_VAL 0xC0ED0000
```

If the magic number is absent, then the last two arguments are not used.

The *data* argument is interpreted by the different file systems.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

The error values given below result from filesystem type independent errors. Each filesystem type may have its own special errors and its own special behavior. See the kernel source code for details.

EPERM The user is not the super-user.

ENODEV *Filesystemtype* not configured in the kernel.

ENOTBLK *Specialfile* is not a block device (if a device was required).

EBUSY *Specialfile* is already mounted. Or, it cannot be remounted read-only, because it still holds files open for writing. Or, it cannot be mounted on *dir* because *dir* is still busy (it is the working directory of some task, the mount point of another device, has open files, etc.).

EINVAL *Specialfile* had an invalid superblock. Or, a remount was attempted, while *specialfile* was not already mounted on *dir*. Or, an umount was attempted, while *dir* was not a mount point.

EFAULT One of the pointer arguments points outside the user address space.

ENOMEM The kernel could not allocate a free page to copy filenames or data into.

ENAMETOOLONG
A pathname was longer than MAXPATHLEN.

ENOENT A pathname was empty or had a nonexistent component.

ENOTDIR The second argument, or a prefix of the first argument, is not a directory.

EACCES A component of a path was not searchable.
Or, mounting a read-only filesystem was attempted without giving the MS_RDONLY flag.
Or, the block device *Specialfile* is located on a filesystem mounted with the MS_NODEV option.

ENXIO The major number of the block device *specialfile* is out of range.

EMFILE (In case no block device is required:) Table of dummy devices is full.

CONFORMING TO

These functions are Linux-specific and should not be used in programs intended to be portable.

SEE ALSO

mount(8), **umount**(8)

TOC

TOC

NAME

mprotect - control allowable accesses to a region of memory

SYNOPSIS

```
#include <sys/mman.h>

int mprotect(const void *addr, size_t len, int prot));
```

DESCRIPTION

mprotect controls how a section of memory may be accessed. If an access is disallowed by the protection given it, the program receives a **SIGSEGV**.

prot is a bitwise-or of the following values:

PROT_NONE The memory cannot be accessed at all.

PROT_READ The memory can be read.

PROT_WRITE The memory can be written to.

PROT_EXEC The memory can contain executing code.

The new protection replaces any existing protection. For example, if the memory had previously been marked **PROT_READ**, and **mprotect** is then called with *prot* **PROT_WRITE**, it will no longer be readable.

RETURN VALUE

On success, **mprotect** returns zero. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL *addr* is not a valid pointer, or not a multiple of PAGESIZE.

EFAULT The memory cannot be accessed.

EACCES The memory cannot be given the specified access. This can happen, for example, if you **mmap**(2) a file to which you have read-only access, then ask **mprotect** to mark it **PROT_WRITE**.

ENOMEM Internal kernel structures could not be allocated.

EXAMPLE

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/mman.h>

#include <limits.h>    /* for PAGESIZE */
#ifndef PAGESIZE
#define PAGESIZE 4096
#endif

int
main(void)
{
    char *p;
    char c;

    /* Allocate a buffer; it will have the default
       protection of PROT_READ|PROT_WRITE. */
    p = malloc(1024+PAGESIZE-1);
```


```
    if (!p) {
        perror("Couldn't malloc(1024)");
        exit(errno);
    }

    /* Align to a multiple of PAGESIZE, assumed to be a power of two */
    p = (char *)(((int) p + PAGESIZE-1) & ~(PAGESIZE-1));

    c = p[666];         /* Read; ok */
    p[666] = 42;        /* Write; ok */

    /* Mark the buffer read-only. */
    if (mprotect(p, 1024, PROT_READ)) {
        perror("Couldn't mprotect");
        exit(errno);
    }

    c = p[666];         /* Read; ok */
    p[666] = 42;        /* Write; program dies on SIGSEGV */

    exit(0);
}
```

CONFORMING TO

SVr4, POSIX.1b (formerly POSIX.4). SVr4 defines an additional error code EAGAIN. The SVr4 error conditions don't map neatly onto Linux's. POSIX.1b says that **mprotect** can be used only on regions of memory obtained from **mmap**(2).

SEE ALSO

mmap(2)

NAME

mremap - re-map a virtual memory address

SYNOPSIS

```
#include <unistd.h>
#include <sys/mman.h>

void * mremap(void * old_address, size_t old_size , size_t
new_size, unsigned long flags));
```

DESCRIPTION

mremap expands (or shrinks) an existing memory mapping, potentially moving it at the same time (controlled by the *flags* argument and the available virtual address space).

old_address is the old address of the virtual memory block that you want to expand (or shrink). Note that *old_address* has to be page aligned. *old_size* is the old size of the virtual memory block. *new_size* is the requested size of the virtual memory block after the resize.

The *flags* argument is a bitmap of flags.

In Linux the memory is divided into pages. A user process has (one or) several linear virtual memory segments. Each virtual memory segment has one or more mappings to real memory pages (in the page table). Each virtual memory segment has its own protection (access rights), which may cause a segmentation violation if the memory is accessed incorrectly (e.g., writing to a read-only segment). Accessing virtual memory outside of the segments will also cause a

segmentation violation.

mremap uses the Linux page table scheme. **mremap** changes the mapping between virtual addresses and memory pages. This can be used to implement a very efficient **realloc**.

FLAGS

MREMAP_MAYMOVE
indicates if the operation should fail, or change the virtual address if the resize cannot be done at the current virtual address.

RETURN VALUE

On success **mremap** returns a pointer to the new virtual memory area. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
An invalid argument was given. Most likely *old_address* was not page aligned.

EFAULT
"Segmentation fault." Some address in the range *old_address* to *old_address+old_size* is an invalid virtual memory address for this process. You can also get EFAULT even if there exist mappings that cover the whole address space requested, but those mappings are

of different types.

EAGAIN
The memory segment is locked and cannot be re-mapped.

ENOMEM
The memory area cannot be expanded at the current virtual address, and the *MREMAP_MAYMOVE* flag is not set in *flags*. Or, there is not enough (virtual) memory available.

CONFORMING TO

This call is Linux-specific, and should not be used in programs intended to be portable. 4.2BSD had a (never actually implemented) **mremap**(2) call with completely different semantics.

SEE ALSO

getpagesize(2), **realloc**(3), **malloc**(3), **brk**(2), **sbrk**(2), **mmap**(2)

Your favorite OS text book for more information on paged memory. (*Modern Operating Systems* by Andrew S. Tannenbaum, *Inside Linux* by Randolf Bentson, *The Design of the UNIX Operating System* by Maurice J. Bach.)

TOC

NAME

msgctl - message control operations

SYNOPSIS

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>

int msgctl ( int msqid, int  cmd, struct msqid_ds *buf )
```

DESCRIPTION

The function performs the control operation specified by *cmd* on the message queue with identifier *msqid*. Legal values for *cmd* are:

IPC_STAT Copy info from the message queue data structure into the structure pointed to by *buf*. The user must have read access privileges on the message queue.

IPC_SET Write the values of some members of the **msqid_ds** structure pointed to by *buf* to the message queue data structure, updating also its **msg_ctime** member. Considered members from the user supplied **struct msqid_ds** pointed to by *buf* are

```
msg_perm.uid
msg_perm.gid
msg_perm.mode  /* only lowest 9-bits */
msg_qbytes
```

The calling process effective user-ID must be one among super-user, creator or owner of the message queue. Only the super-user can raise the **msg_qbytes** value beyond the system parameter **MSGMNB**.

IPC_RMID Remove immediately the message queue and its data structures awakening all waiting reader and writer processes (with an error return and **errno** set to **EIDRM**). The calling process effective user-ID must be one among super-user, creator or owner of the message queue.

RETURN VALUE

If successful, the return value will be **0**, otherwise **-1** with **errno** indicating the error.

ERRORS

For a failing return, **errno** will be set to one among the following values:

EACCES The argument *cmd* is equal to **IPC_STAT** but the calling process has no read access permissions on the message queue *msqid*.

EFAULT The argument *cmd* has value **IPC_SET** or **IPC_STAT** but the address pointed to by *buf* isn't accessible.

EIDRM The message queue was removed.

EINVAL Invalid value for *cmd* or *msqid*.

EPERM The argument *cmd* has value **IPC_SET** or **IPC_RMID** but the calling process effective user-ID has insufficient privileges to execute the command.

Note this is also the case of a non super-user process trying to increase the **msg_qbytes** value beyond the value specified by the system parameter **MSGMNB**.

NOTES

The **IPC_INFO**, **MSG_STAT** and **MSG_INFO** control calls are used by the **ipcs**(8) program to provide information on allocated resources. In the future these can be modified as needed or moved to a proc file system interface.

CONFORMING TO

SVr4, SVID. SVID dies not document the EIDRM error condition.

SEE ALSO

ipc(5), **msgget**(2), **msgsnd**(2), **msgrcv**(2).

TOC

TOC

NAME

msgget - get a message queue identifier

SYNOPSIS

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>

int msgget ( key_t key, int msgflg )
```

DESCRIPTION

The function returns the message queue identifier associated to the value of the *key* argument. A new message queue is created if *key* has value **IPC_PRIVATE** or *key* isn't **IPC_PRIVATE**, no existing message queue is associated to *key*, and **IPC_CREAT** is asserted in *msgflg* (i.e. *msgflg&IPC_CREAT* is nonzero). The presence in *msgflg* of the fields **IPC_CREAT** and **IPC_EXCL** plays the same role, with respect to the existence of the message queue, as the presence of **O_CREAT** and **O_EXCL** in the mode argument of the **open**(2) system call: i.e. the **msgget** function fails if *msgflg* asserts both **IPC_CREAT** and **IPC_EXCL** and a message queue already exists for *key*.

Upon creation, the lower 9 bits of the argument *msgflg* define the access permissions of the message queue. These permission bits have the same format and semantics as the access permissions parameter in **open**(2) or **creat**(2) system calls. (The execute permissions are not used.)

Furthermore, while creating, the system call initializes the

system message queue data structure **msqid_ds** as follows:

msg_perm.cuid and **msg_perm.uid** are set to the effective user-ID of the calling process.

msg_perm.cgid and **msg_perm.gid** are set to the effective group-ID of the calling process.

The lowest order 9 bits of **msg_perm.mode** are set to the lowest order 9 bit of *msgflg*.

msg_qnum, **msg_lspid**, **msg_lrpid**, **msg_stime** and **msg_rtime** are set to 0.

msg_ctime is set to the current time.

msg_qbytes is set to the system limit **MSGMNB**.

If the message queue already exists the access permissions are verified, and a check is made to see if it is marked for destruction.

RETURN VALUE

If successful, the return value will be the message queue identifier (a nonnegative integer), otherwise **-1** with **errno** indicating the error.

ERRORS

For a failing return, **errno** will be set to one among the following values:

EACCES A message queue exists for *key*, but the calling process has no access permissions to the queue.

EEXIST A message queue exists for **key** and *msgflg* was asserting both **IPC_CREAT** and **IPC_EXCL**.

EIDRM The message queue is marked for removal.

ENOENT No message queue exists for *key* and *msgflg* wasn't asserting **IPC_CREAT**.

ENOMEM A message queue has to be created but the system has not enough memory for the new data structure.

ENOSPC A message queue has to be created but the system limit for the maximum number of message queues (**MSGMNI**) would be exceeded.

NOTES

IPC_PRIVATE isn't a flag field but a **key_t** type. If this special value is used for *key*, the system call ignores everything but the lowest order 9 bits of *msgflg* and creates a new message queue (on success).

The following is a system limit on message queue resources affecting a **msgget** call:

MSGMNI System wide maximum number of message queues: policy dependent.

BUGS

Use of **IPC_PRIVATE** does not actually prohibit other processes from getting access to the allocated message queue.

As for the files, there is currently no intrinsic way for a process to ensure exclusive access to a message queue. Asserting both **IPC_CREAT** and **IPC_EXCL** in *msgflg* only ensures (on success) that a new message queue will be created, it doesn't imply exclusive access to the message queue.

CONFORMING TO

SVr4, SVID. SVr4 does not document the EIDRM error code.

SEE ALSO

ftok(3), **ipc**(5), **msgctl**(2), **msgsnd**(2), **msgrcv**(2).


TOC

NAME

msgop - message operations

SYNOPSIS

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>

int msgsnd ( int msqid, struct msgbuf *msgp, int msgsz,  int
msgflg )

int msgrcv ( int msqid, struct msgbuf *msgp, int msgsz, long
msgtyp, int msgflg )
```

DESCRIPTION

To send or receive a message, the calling process allocates a structure that looks like the following

```
     struct msgbuf {
          long mtype;     /* message type, must be > 0 */
          char mtext[1];  /* message data */
     };
```

but with an array **mtext** of size *msgsz*, a non-negative integer value. The structure member **mtype** must have a strictly positive integer value that can be used by the receiving process for message selection (see the section about **msgrcv**).

The calling process must have write access permissions to send and read access permissions to receive a message on the

queue.

The **msgsnd** system call enqueue a copy of the message pointed to by the *msgp* argument on the message queue whose identifier is specified by the value of the *msqid* argument.

The argument *msgflg* specifies the system call behaviour if enqueuing the new message will require more than **msg_qbytes** in the queue. Asserting **IPC_NOWAIT** the message will not be sent and the system call fails returning with **errno** set to **EAGAIN**. Otherwise the process is suspended until the condition for the suspension no longer exists (in which case the message is sent and the system call succeeds), or the queue is removed (in which case the system call fails with **errno** set to **EIDRM**), or the process receives a signal that has to be caught (in which case the system call fails with **errno** set to **EINTR**).

Upon successful completion the message queue data structure is updated as follows:

msg_lspid is set to the process-ID of the calling process.

msg_qnum is incremented by 1.

msg_stime is set to the current time.

The system call **msgrcv** reads a message from the message queue specified by *msqid* into the **msgbuf** pointed to by the *msgp* argument removing from the queue, on success, the read message.

The argument *msgsz* specifies the maximum size in bytes for the member **mtext** of the structure pointed to by the *msgp* argument. If the message text has length greater than *msgsz*, then if the *msgflg* argument asserts **MSG_NOERROR**, the message text will be truncated (and the truncated part will be lost), otherwise the message isn't removed from the queue and the system call fails returning with **errno** set to **E2BIG**.

The argument *msgtyp* specifies the type of message requested as follows:

If *msgtyp* is **0**, then the message on the queue's front is read.

If *msgtyp* is greater than **0**, then the first message on the queue of type *msgtyp* is read if **MSG_EXCEPT** isn't asserted by the *msgflg* argument, otherwise the first message on the queue of type not equal to *msgtyp* will be read.

If *msgtyp* is less than **0**, then the first message on the queue with the lowest type less than or equal to the absolute value of *msgtyp* will be read.

The *msgflg* argument asserts none, one or more (or-ing them) among the following flags:

IPC_NOWAIT For immediate return if no message of the requested type is on the queue. The system call fails with errno set to **ENOMSG**.

MSG_EXCEPT Used with *msgtyp* greater than **0** to read the first message on the queue with message type that differs from *msgtyp*.

MSG_NOERROR To truncate the message text if longer than *msgsz* bytes.

If no message of the requested type is available and **IPC_NOWAIT** isn't asserted in *msgflg*, the calling process is blocked until one of the following conditions occurs:

A message of the desired type is placed on the queue.

The message queue is removed from the system. In such a case the system call fails with **errno** set to **EIDRM**.

The calling process receives a signal that has to be caught. In such a case the system call fails with **errno** set to **EINTR**.

Upon successful completion the message queue data structure is updated as follows:

msg_lrpid is set to the process-ID of the calling process.

msg_qnum is decremented by 1.

msg_rtime is set to the current time.

RETURN VALUE

On a failure both functions return **-1** with **errno** indicating the error, otherwise **msgsnd** returns **0** and **msgrvc** returns the number of bytes actually copied into the **mtext** array.

ERRORS

When **msgsnd** fails, at return **errno** will be set to one among the following values:

EAGAIN The message can't be sent due to the **msg_qbytes** limit for the queue and **IPC_NOWAIT** was asserted in *mgsflg*.

EACCES The calling process has no write access permissions on the message queue.

EFAULT The address pointed to by *msgp* isn't accessible.

EIDRM The message queue was removed.

EINTR Sleeping on a full message queue condition, the process received a signal that had to be caught.

EINVAL Invalid *msqid* value, or nonpositive *mtype* value, or invalid *msgsz* value (less than 0 or greater than the system value **MSGMAX**).

ENOMEM The system has not enough memory to make a copy of the supplied **msgbuf**.

When **msgrcv** fails, at return **errno** will be set to one among the following values:

E2BIG The message text length is greater than *msgsz* and **MSG_NOERROR** isn't asserted in *msgflg*.

EACCES The calling process has no read access permissions on the message queue.

EFAULT The address pointed to by *msgp* isn't accessible.

EIDRM While the process was sleeping to receive a message, the message queue was removed.

EINTR While the process was sleeping to receive a message, the process received a signal that had to be caught.

EINVAL Illegal *msgqid* value, or *msgsz* less than **0**.

ENOMSG **IPC_NOWAIT** was asserted in *msgflg* and no message of the requested type existed on the message queue.

NOTES

The followings are system limits affecting a **msgsnd** system call:

MSGMAX Maximum size for a message text: the implementation set this value to 4080 bytes.

MSGMNB Default maximum size in bytes of a message queue: policy dependent. The super-user can increase the size of a message queue beyond **MSGMNB** by a **msgctl** system call.

The implementation has no intrinsic limits for the system wide maximum number of message headers (**MSGTQL**) and for the system wide maximum size in bytes of the message pool (**MSGPOOL**).

CONFORMING TO

SVr4, SVID.

SEE ALSO

ipc(5), **msgctl**(2), **msgget**(2), **msgrcv**(2), **msgsnd**(2).

TOC

NAME

msync - synchronize a file with a memory map

SYNOPSIS

```
#include <unistd.h>
#include <sys/mman.h>

#ifdef _POSIX_MAPPED_FILES
#ifdef _POSIX_SYNCHRONIZED_IO

int msync(const void *start, size_t length, int flags

#endif
#endif
```

DESCRIPTION

msync flushes changes made to the in-core copy of a file that was mapped into memory using **mmap**(2) back to disk. Without use of this call there is no guarantee that changes are written back before **munmap**(2) is called. To be more precise, the part of the file that corresponds to the memory area starting at *start* and having length *length* is updated. The *flags* argument may have the bits MS_ASYNC, MS_SYNC and MS_INVALIDATE set, but not both MS_ASYNC and MS_SYNC. MS_ASYNC specifies that an update be scheduled, but the call returns immediately. MS_SYNC asks for an update and waits for it to complete. MS_INVALIDATE asks to invalidate other mappings of the same file (so that they can be updated with the fresh values just written).

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
start is not a multiple of PAGESIZE, or any bit other than MS_ASYNC | MS_INVALIDATE | MS_SYNC is set in *flags*.

EFAULT
The indicated memory (or part of it) was not mapped.

CONFORMING TO

POSIX.1b (formerly POSIX.4)

SEE ALSO

mmap(2), B.O. Gallmeister, POSIX.4, O'Reilly, pp. 128-129 and 389-391.

TOC

NAME

munlock - reenable paging for some parts of memory

SYNOPSIS

#include <sys/mman.h>

int munlock(const void * *addr*, **size_t** *len*));

DESCRIPTION

munlock reenables paging for the memory in the range starting at *addr* with length *len* bytes. All pages which contain a part of the specified memory range can after calling **munlock** be moved to external swap space again by the kernel.

Memory locks do not stack, i.e., pages which have been locked several times by calls to **mlock** or **mlockall** will be unlocked by a single call to **munlock** for the corresponding range or by **munlockall**. Pages which are mapped to several locations or by several processes stay locked into RAM as long as they are locked at least at one location or by at least one process.

On POSIX systems on which **mlock** and **munlock** are available, **_POSIX_MEMLOCK_RANGE** is defined in <unistd.h> and the value **PAGESIZE** from <limits.h> indicates the number of bytes per page.

RETURN VALUE

On success, **munlock** returns zero. On error, -1 is returned, *errno* is set appropriately, and no changes are made to any locks in the address space of the process.

ERRORS

ENOMEM Some of the specified address range does not correspond to mapped pages in the address space of the process.

EINVAL *len* was not a positive number.

CONFORMING TO

POSIX.1b, SVr4

SEE ALSO

mlock(2), **mlockall**(2), and **munlockall**(2).

NAME

munlockall - reenable paging for calling process

SYNOPSIS

#include <sys/mman.h>

int munlockall(void);

DESCRIPTION

munlockall reenables paging for all pages mapped into the address space of the calling process.

Memory locks do not stack, i.e., pages which have been locked several times by calls to **mlock** or **mlockall** will be unlocked by a single call to **munlockall**. Pages which are mapped to several locations or by several processes stay locked into RAM as long as they are locked at least at one location or by at least one process.

On POSIX systems on which **mlockall** and **munlockall** are available, **_POSIX_MEMLOCK** is defined in *<unistd.h>* .

RETURN VALUE

On success, **munlockall** returns zero. On error, -1 is returned and *errno* is set appropriately.

CONFORMING TO

POSIX.1b, SVr4

SEE ALSO

mlockall(2), **mlock**(2), and **munlock**(2).

TOC

NAME

nanosleep - pause execution for a specified time

SYNOPSIS

#include <time.h>

int nanosleep(const struct timespec **req*, struct timespec **rem*));

DESCRIPTION

nanosleep delays the execution of the program for at least the time specified in **req*. The function can return earlier if a signal has been delivered to the process. In this case, it returns -1, sets *errno* to **EINTR**, and writes the remaining time into the structure pointed to by *rem* unless *rem* is **NULL**. The value of **rem* can then be used to call **nanosleep** again and complete the specified pause.

The structure *timespec* is used to specify intervals of time with nanosecond precision. It is specified in *<time.h>* and has the form

```
struct timespec
{
        time_t  tv_sec;         /* seconds */
        long    tv_nsec;        /* nanoseconds */
};
```

The value of the nanoseconds field must be in the range 0 to 999 999 999.

Compared to **sleep**(3) and **usleep**(3), **nanosleep** has the advantage of not affecting any signals, it is standardized by POSIX, it provides higher timing resolution, and it allows to continue a sleep that has been interrupted by a signal more easily.

ERRORS

In case of an error or exception, the **nanosleep** system call returns -1 instead of 0 and sets *errno* to one of the following values:

EINTR The pause has been interrupted by a non-blocked signal that was delivered to the process. The remaining sleep time has been written into **rem* so that the process can easily call **nanosleep** again and continue with the pause.

EINVAL The value in the *tv_nsec* field was not in the range 0 to 999 999 999 or *tv_sec* was negative.

BUGS

The current implementation of **nanosleep** is based on the normal kernel timer mechanism, which has a resolution of 1/*HZ* s (i.e, 10 ms on Linux/i386 and 1 ms on Linux/Alpha). Therefore, **nanosleep** pauses always for at least the specified time, however it can take up to 10 ms longer than specified until the process becomes runnable again. For the same reason, the value returned in case of a delivered signal in **rem* is usually rounded to the next larger multiple of 1/*HZ* s.

As some applications require much more precise pauses (e.g., in order to control some time-critical hardware), **nanosleep** is also capable of short high-precision pauses. If the process is scheduled under a real-time policy like *SCHED_FIFO*

or *SCHED_RR*, then pauses of up to 2 ms will be performed as busy waits with microsecond precision.

CONFORMING TO

POSIX.1b (formerly POSIX.4).

SEE ALSO

sleep(3), **usleep**(3), **sched_setscheduler**(2), and **timer_create**(2).

TOC

NAME

nfsservctl - syscall interface to kernel nfs daemon

SYNOPSIS

```
#include <linux/nfsd/syscall.h>

nfsservctl(int cmd, struct nfsctl_arg *argp, union
nfsctl_res *resp
```

DESCRIPTION

```
/*
 * These are the commands understood by nfsctl().
 */
#define NFSCTL_SVC              0       /* This is a server process. */
#define NFSCTL_ADDCLIENT        1       /* Add an NFS client. */
#define NFSCTL_DELCLIENT        2       /* Remove an NFS client. */
#define NFSCTL_EXPORT           3       /* export a file system. */
#define NFSCTL_UNEXPORT         4       /* unexport a file system. */
#define NFSCTL_UGIDUPDATE       5       /* update a client's uid/gid map. */
#define NFSCTL_GETFH            6       /* get an fh (used by mountd) */

struct nfsctl_arg {
        int                     ca_version;     /* safeguard */
        union {
                struct nfsctl_svc       u_svc;
                struct nfsctl_client    u_client;
                struct nfsctl_export    u_export;
                struct nfsctl_uidmap    u_umap;
                struct nfsctl_fhparm    u_getfh;
                unsigned int            u_debug;
        } u;
}

union nfsctl_res {
        struct knfs_fh          cr_getfh;
        unsigned int            cr_debug;
};
```

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

CONFORMING TO

This call is Linux-specific.

NAME

nice - change process priority

SYNOPSIS

#include <unistd.h>

int nice(int *inc*));

DESCRIPTION

nice adds *inc* to the priority for the calling pid. Only the superuser may specify a negative increment, or priority increase.

Note that internally, a higher number is a higher priority. Do not confuse this with the priority scheme as used by the **nice** interface.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM A non-super user attempts to do a priority increase, a numerical decrease, by supplying a negative *inc*.

CONFORMING TO

SVr4, SVID EXT, AT&T, X/OPEN, BSD 4.3. SVr4 documents an additional EINVAL error code.

SEE ALSO

nice(1), **setpriority**(2), **fork**(2),

NAME

oldfstat, oldlstat, oldstat, oldolduname, olduname - obsolete system calls

SYNOPSIS

Obsolete system calls.

DESCRIPTION

The Linux 2.0 kernel implements these calls to support old executables. These calls return structures which have grown since their first implementation, but old executables must continue to receive old smaller structures.

Current executables should be linked with current libraries and never use these calls.

CONFORMING TO

These calls are unique to Linux and should not be used at all in new programs.

SEE ALSO

fstat(2), **lstat**(2), **stat**(2), **uname**(2), **undocumented**(2), **unimplemented**(2)

NAME

open, creat - open and possibly create a file or device

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags));
int open(const char *pathname, int flags, mode_t mode
int creat(const char *pathname, mode_t mode));
```

DESCRIPTION

open attempts to open a file and return a file descriptor (a small, non-negative integer for use in **read**, **write**, etc.)

flags is one of **O_RDONLY**, **O_WRONLY** or **O_RDWR** which request opening the file read-only, write-only or read/write, respectively.

flags may also be bitwise-*or*'d with one or more of the following:

O_CREAT If the file does not exist it will be created.

O_EXCL When used with **O_CREAT**, if the file already exists it is an error and the **open** will fail. **O_EXCL** is broken on NFS file systems, programs which rely on it for performing locking tasks will contain a race condition. The solution for performing atomic file locking using a lockfile is to create a unique file

on the same fs (e.g., incorporating hostname and pid), use **link**(2) to make a link to the lockfile and use **stat**(2) on the unique file to check if its link count has increased to 2. Do not use the return value of the **link()** call.

O_NOCTTY
If *pathname* refers to a terminal device - see **tty**(4) - it will not become the process's controlling terminal even if the process does not have one.

O_TRUNC If the file already exists it will be truncated.

O_APPEND
The file is opened in append mode. Initially, and before each **write**, the file pointer is positioned at the end of the file, as if with **lseek**. **O_APPEND** may lead to corrupted files on NFS file systems if more than one process appends data to a file at once. This is because NFS does not support appending to a file, so the client kernel has to simulate it, which can't be done without a race condition.

O_NONBLOCK or **O_NDELAY**
The file is opened in non-blocking mode. Neither the **open** nor any subsequent operations on the file descriptor which is returned will cause the calling process to wait.

O_SYNC The file is opened for synchronous I/O. Any **write**s on the resulting file descriptor will block the calling process until the data has been physically written to the underlying hardware. *See RESTRICTIONS below, though*.

Some of these optional flags can be altered using **fcntl** after the file has been opened.

mode specifies the permissions to use if a new file is created. It is modified by the process's **umask** in the usual way: the permissions of the created file are **(mode & ~umask)**.

The following symbolic constants are provided for *mode*:

S_IRWXU

00700 user (file owner) has read, write and execute permission

S_IRUSR (S_IREAD)
00400 user has read permission

S_IWUSR (S_IWRITE)
00200 user has write permission

S_IXUSR (S_IEXEC)
00100 user has execute permission

S_IRWXG
00070 group has read, write and execute permission

S_IRGRP
00040 group has read permission

S_IWGRP
00020 group has write permission

S_IXGRP
00010 group has execute permission

S_IRWXO
00007 others have read, write and execute permission

S_IROTH
00004 others have read permission

S_IWOTH
00002 others have write permisson

S_IXOTH
00001 others have execute permission

mode should always be specified when **O_CREAT** is in the *flags*, and is ignored otherwise.

creat is equivalent to **open** with *flags* equal to **O_CREAT|O_WRONLY|O_TRUNC**.

RETURN VALUE

open and **creat** return the new file descriptor, or -1 if an error occurred (in which case, *errno* is set appropriately). Note that **open** can open device special files, but **creat** cannot create them - use **mknod**(2) instead.

On NFS file systems with UID mapping enabled, **open** may return a file descriptor but e.g. **read**(2) requests are denied with **EACCES**. This is because the client performs **open** by checking the permissions, but UID mapping is performed by the server upon read and write requests.

ERRORS

EEXIST *pathname* already exists and **O_CREAT** and **O_EXCL** were used.

EISDIR *pathname* refers to a directory and the access requested involved writing.

ETXTBSY *pathname* refers to an executable image which is currently being executed and write access was requested.

EFAULT *pathname* points outside your accessible address space.

EACCES The requested access to the file is not allowed, or one of the directories in *pathname* did not allow search (execute) permission.

ENAMETOOLONG
pathname was too long.

ENOENT A directory component in *pathname* does not exist or is a dangling symbolic link.

ENOTDIR A component used as a directory in *pathname* is not, in fact, a directory.

EMFILE The process already has the maximum number of files open.

ENFILE The limit on the total number of files open on the system has been reached.

ENOMEM Insufficient kernel memory was available.

EROFS *pathname* refers to a file on a read-only filesystem and write access was requested.

ELOOP Too many symbolic links were encountered in resolving *pathname*.

ENOSPC *pathname* was to be created but the device containing *pathname* has no room for the new file.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, BSD 4.3

RESTRICTIONS

There are many infelicities in the protocol underlying NFS, affecting amongst others **O_SYNC** and **O_NDELAY**.

SEE ALSO

read(2), **write**(2), **fcntl**(2), **unlink**(2), **mknod**(2), **stat**(2), **mount**(2), **socket**(2), **socket**(2), **link**(2).

TOC

TOC

NAME

outb, outw, outl, outsb, outsw, outsl - port output
inb, inw, inl, insb, insw, insl - port input
outb_p, outw_p, outl_p, inb_p, inw_p, inl_p - paused I/O

DESCRIPTION

This family of functions is used to do low level port input and output. They are primarily designed for internal kernel use, but can be used from user space, given the following information *in addition* to that given in *outb(9)*

You compile with **-O** or **-O2** or similar. The functions are defined as inline macros, and will not be substituted in without optimization enabled, causing unresolved references at link time.

You use *ioperm(2)* or alternatively *iopl(2)* to tell the kernel to allow the user space application to access the I/O ports in question. Failure to do this will cause the application to receive a segmentation fault.

CONFORMING TO

outb and friends are hardware specific. The *port* and *value* arguments are in the opposite order to most DOS implementations.

SEE ALSO

outb(9), **ioperm**(2), **iopl**(2)

NAME

pause - wait for signal

SYNOPSIS

```
#include <unistd.h>

int pause(void);
```

DESCRIPTION

The **pause** system call causes the invoking process to sleep until a signal is received.

RETURN VALUE

pause always returns -1, and *errno* is set to **ERESTARTNOHAND**.

ERRORS

EINTR
signal was received.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, BSD 4.3

SEE ALSO

kill(2), **select**(2), **signal**(2)

TOC

NAME

personality - set the process execution domain

SYNOPSIS

int personality(unsigned long *persona*));

DESCRIPTION

Linux supports different execution domains, or personalities, for each process. Among other things, execution domains tell Linux how to map signal numbers into signal actions. The execution domain system allows Linux to provide limited support for binaries compiled under other Unix-like operating systems.

personality will make the execution domain referenced by *persona* the new execution domain of the current process.

RETURN VALUE

On success, *persona* is made the new execution domain and the previous *persona* is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL *persona* does not refer to a valid execution domain.

FILES

/usr/include/linux/personality.h

CONFORMING TO

personality is Linux-specific and should not be used in programs intended to be portable.

NAME

pipe - create pipe

SYNOPSIS

```
#include <unistd.h>

int pipe(int filedes[2]);
```

DESCRIPTION

pipe creates a pair of file descriptors, pointing to a pipe inode, and places them in the array pointed to by *filedes*. *filedes*[**0**] is for reading, *filedes*[**1**] is for writing.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EMFILE Too many file descriptors are in use by the process.

ENFILE The system file table is full.

EFAULT *filedes* is not valid.

CONFORMING TO

SVr4, SVID, AT&T, POSIX, X/OPEN, BSD 4.3

SEE ALSO

read(2), **write**(2), **fork**(2),

NAME

poll - wait for some event on a file descriptor

SYNOPSIS

```
#include <sys/poll.h>

int poll(struct pollfd *ufds, unsigned int nfds, int timeout
```

DESCRIPTION

poll is a variation on the theme of **select**. It specifies an array of *nfds* structures of type

```
        struct pollfd {
                int fd;           /* file descriptor */
                short events;     /* requested events */
                short revents;    /* returned events */
        };
```

and a *timeout* in milliseconds. A negative value means infinite timeout. The field *fd* contains a file descriptor for an open file. The field *events* is an input parameter, a bitmask specifying the events the application is interested in. The field *revents* is an output parameter, filled by the kernel with the events that actually occurred, either of the type requested, or of one of the types **POLLERR** or **POLLHUP** or **POLLNVAL**. (These three bits are meaningless in the *events* field, and will be set in the *revents* field whenever the corresponding condition is true.) If none of the events requested (and no error) has occurred for any of the file descriptors, the kernel waits for *timeout* milliseconds for one of these events to occur. The following possible bits in these masks are defined in <sys/poll.h>

```
    #define POLLIN      0x0001    /* There is data to read */
    #define POLLPRI     0x0002    /* There is urgent data to read */
```


```
    #define POLLOUT     0x0004    /* Writing now will not block */
    #define POLLERR     0x0008    /* Error condition */
    #define POLLHUP     0x0010    /* Hung up */
    #define POLLNVAL    0x0020    /* Invalid request: fd not open */
```

In <asm/poll.h> also the values **POLLRDNORM**, **POLLRDBAND**, **POLLWRNORM**, **POLLWRBAND** and **POLLMSG** are defined.

RETURN VALUE

On success, a positive number is returned, where the number returned is the number of structures which have non-zero *revents* fields (in other words, those descriptors with events or errors reported). A value of 0 indicates that the call timed out and no file descriptors have been selected. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

ENOMEM
There was no space to allocate file descriptor tables.

EFAULT
The array given as argument was not contained in the calling program's address space.

EINTR
A signal occurred before any requested event.

CONFORMING TO

XPG4-UNIX.

AVAILABILITY

The poll() systemcall was introduced in Linux 2.1.23. The poll() library call was introduced in libc 5.4.28 (and provides emulation using select if your kernel does not have a poll syscall).

SEE ALSO

select(2)

TOC

NAME

prctl - operations on a process

SYNOPSIS

#include <linux/prctl.h>

int prctl(int *option*, **unsigned long** *arg2*, **unsigned long** *arg3*, **unsigned long** *arg4*, **unsigned long** *arg5*));

DESCRIPTION

prctl is called with a first argument describing what to do (with values defined in <*linux/prctl.h*>), and further parameters with a significance depending on the first one. At present the only option value defined is **PR_SET_PDEATHSIG**, and the corresponding call sets the parent process death signal of the current process to *arg2* (either a signal value in the range 1..maxsig, or 0 to clear). This is the signal that the current process will get when its parent dies. This value is cleared upon a fork().

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
The value of *option* is not recognized, or it is **PR_SET_PDEATHSIG** and *arg2* is not zero or a signal number.

CONFORMING TO

This call is Linux-specific. IRIX has a ptctl system call (also introduced in Linux 2.1.44 as irix_prctl on the MIPS architecture), with prototype

ptrdiff_t prctl(int *option*, **int** *arg2*, **int** *arg3*

and options to get the maximum number of processes per user, get the maximum number of processors the calling process can use, find out whether a specified process is currently blocked, get or set the maximum stack size, etc., etc.

AVAILABILITY

The ptctl() systemcall was introduced in Linux 2.1.57. There is no ptctl() library call as yet.

SEE ALSO

signal(2)

NAME

ptrace - process trace

SYNOPSIS

#include <sys/ptrace.h>

int ptrace(int *request*, **int** *pid*, **int** *addr*

DESCRIPTION

Ptrace provides a means by which a parent process may control the execution of a child process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging. A traced process runs until a signal occurs. Then it stops and the parent will be notified with **wait**(2). When the process is in the stopped state, its memory can be read and written. The parent can also cause the child to continue execution, with optional ignoring the signal which caused stopping.

The value of the *request* argument determines the precise action of the system call:

PTRACE_TRACEME
This process is to be traced by its parent. The parent should be expecting to trace the child.

PTRACE_PEEKTEXT, PTRACE_PEEKDATA
Read word at location *addr*.

PTRACE_PEEKUSR
Read word at location *addr* in the **USER** area.

PTRACE_POKETEXT, PTRACE_POKEDATA
Write word at location *addr*.

PTRACE_POKEUSR
Write word at location *addr* in the **USER** area.

PTRACE_SYSCALL, PTRACE_CONT
Restart after signal.

PTRACE_KILL
Send the child a **SIGKILL** to make it exit.

PTRACE_SINGLESTEP
Set the trap flag for single stepping.

PTRACE_ATTACH
Attach to the process specified in *pid*.

PTRACE_DETACH
Detach a process that was previously attached.

NOTES

init, the process with process ID 1, may not use this function.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM The specified process (i.e., **init**), cannot be traced, or is already being traced.

ESRCH The specified process does not exist.

EIO *Request* is not valid.

CONFORMING TO

SVr4, SVID EXT, AT&T, X/OPEN, BSD 4.3

SEE ALSO

gdb(1), **exec**(3), **signal**(2), **wait**(2)

TOC

NAME

quotactl - manipulate disk quotas

SYNOPSIS

```
#include <sys/types.h>
#include <sys/quota.h>

int quotactl (int cmd, const char *special, int id , caddr_t
addr));

#include <linux/unistd.h>

_syscall4(int, quotactl, int, cmd, const char *,  special  ,
int, id, caddr_t, addr));
```

DESCRIPTION

The quota system defines for each user and/or group a soft limit and a hard limit bounding the amount of disk space that can be used on a given file system. The hard limit cannot be crossed. The soft limit can be crossed, but warnings will ensue. Moreover, the user cannot be above the soft limit for more than one week (by default) at a time: after this week the soft limit counts as hard limit.

The **quotactl** system call manipulates these quota. Its first argument is of the form *QCMD*(**subcmd**,**type**) where *type* is either **USRQUOTA** or **GRPQUOTA** (for user quota and group quota, respectively), and *subcmd* is described below.

The second argument *special* is the block special device these quota apply to. It must be mounted.

The third argument *id* is the user or group id these quota apply to (when relevant).

The fourth argument *addr* is the address of a data structure, depending on the command.

The *subcmd* is one of

Q_QUOTAON Enable quotas. The *addr* argument is the pathname of the file containing the quotas for the filesystem.

Q_QUOTAOFF Disable quotas.

Q_GETQUOTA Get limits and current usage of disk space. The *addr* argument is a pointer to a dqblk structure (defined in <*sys/quota.h*>).

Q_SETQUOTA Set limits and current usage; *addr* is as before.

Q_SETQLIM Set limits; *addr* is as before.

Q_SETUSE Set usage.

Q_SYNC Sync disk copy of a filesystems quotas.

Q_GETSTATS Get collected stats.

RETURN VALUE

On success, **quotactl** returns 0. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

ENOPKG The kernel was compiled without quota support.

TOC

EFAULT Bad *addr* value.

EINVAL *type* is not a known quota type. Or, *special* could not be found.

ENOTBLK *special* is not a block special device.

ENODEV *special* cannot be found in the mount table.

EACCES The quota file is not an ordinary file.

EIO Cannot read or write the quota file.

EMFILE Too many open files: cannot open quota file.

EBUSY **Q_QUOTAON** was asked, but quota were enabled already.

ESRCH **Q_GETQUOTA** or **Q_SETQUOTA** or **Q_SETUSE** or **Q_SETQLIM** was asked for a file system that didn't have quota enabled.

EPERM The process was not root (for the file system), and **Q_GETQUOTA** was asked for another *id* than that of the process itself, or anything other than **Q_GETSTATS** or **Q_SYNC** was asked.

CONFORMING TO

BSD

SEE ALSO

quota(1), **getrlimit**(2), **setrlimit**(2), **ulimit**(2), **quotacheck**(8), **quotaon**(8)

TOC

NAME

read - read from a file descriptor

SYNOPSIS

#include <unistd.h>

ssize_t read(int *fd*, **void ***buf*, **size_t** *count*

DESCRIPTION

read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

If *count* is zero, **read()** returns zero and has no other results. If *count* is greater than SSIZE_MAX, the result is unspecified.

RETURN VALUE

On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. It is not an error if this number is smaller than the number of bytes requested; this may happen for example because fewer bytes are actually available right now (maybe because we were close to end-of-file, or because we are reading from a pipe, or from a terminal), or because **read()** was interrupted by a signal. On error, -1 is returned, and *errno* is set appropriately. In this case it is

left unspecified whether the file position (if any) changes.

ERRORS

EINTR The call was interrupted by a signal before any data was read.

EAGAIN Non-blocking I/O has been selected using **O_NONBLOCK** and no data was immediately available for reading.

EIO I/O error. This will happen for example when the process is in a background process group, tries to read from its controlling tty, and either it is ignoring or blocking SIGTTIN or its process group is orphaned. It may also occur when there is a low-level I/O error while reading from a disk or tape.

EISDIR *fd* refers to a directory.

EBADF *fd* is not a valid file descriptor or is not open for reading.

EINVAL *fd* is attached to an object which is unsuitable for reading.

EFAULT *buf* is outside your accessible address space.

Other errors may occur, depending on the object connected to *fd*. POSIX allows a **read** that is interrupted after reading some data to return -1 (with *errno* set to EINTR) or to return the number of bytes already read.

CONFORMING TO

SVr4, SVID, AT&T, POSIX, X/OPEN, BSD 4.3

RESTRICTIONS

On NFS file systems, reading small amounts of data will only update the time stamp the first time, subsequent calls may not do so. This is caused by client side attribute caching, because most if not all NFS clients leave atime updates to the server and client side reads satisfied from the client's cache will not cause atime updates on the server as there are no server side reads. UNIX semantics can be obtained by disabling client side attribute caching, but in most situations this will substantially increase server load and decrease performance.

SEE ALSO

readdir(2), **write**(2), **write**(2), **lseek**(2), **select**(2), **readlink**(2), **ioctl**(2),


TOC

NAME

readdir - read directory entry

SYNOPSIS

```
#include <unistd.h>
#include <linux/dirent.h>
#include <linux/unistd.h>

_syscall3(int, readdir, uint, fd, struct dirent

int readdir(unsigned int fd, struct dirent *dirp, unsigned int count
```

DESCRIPTION

This is not the function you are interested in. Look at **readdir**(3) for the POSIX conforming C library interface. This page documents the bare kernel system call interface, which can change, and which is superseded by **getdents**(2).

readdir reads one *dirent* structure from the directory pointed at by *fd* into the memory area pointed to by *dirp*. The parameter *count* is ignored; at most one dirent structure is read.

The *dirent* structure is declared as follows:

```
struct dirent
{
    long d_ino;                 /* inode number */
    off_t d_off;                /* offset to this dirent */
    unsigned short d_reclen;    /* length of this d_name */
    char d_name [NAME_MAX+1];   /* file name (null-terminated) */
}
```

d_ino is an inode number. *d_off* is the distance from the

start of the directory to this *dirent*. *d_reclen* is the size of *d_name*, not counting the null terminator. *d_name* is a null-terminated file name.

RETURN VALUE

On success, 1 is returned. On end of directory, 0 is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF
Invalid file descriptor *fd*.

EFAULT
Argument points outside the calling process's address space.

EINVAL
Result buffer is too small.

ENOENT
No such directory.

ENOTDIR
File descriptor does not refer to a directory.

CONFORMING TO

This system call is Linux specific.

SEE ALSO

getdents(2), **readdir**(3)

NAME

readlink - read value of a symbolic link

SYNOPSIS

#include <unistd.h>

int readlink(const char *_path_, char *_buf_, size_t _bufsiz_

DESCRIPTION

readlink places the contents of the symbolic link *path* in the buffer *buf*, which has size *bufsiz*. **readlink** does not append a **NUL** character to *buf*. It will truncate the contents (to a length of *bufsiz* characters), in case the buffer is too small to hold all of the contents.

RETURN VALUES

The call returns the count of characters placed in the buffer if it succeeds, or a -1 if an error occurs, placing the error code in *errno*.

ERRORS

ENOTDIR A component of the path prefix is not a directory.

EINVAL *bufsiz* is not positive.

ENAMETOOLONG
A pathname, or a component of a pathname, was too long.

ENOENT The named file does not exist.

EACCES Search permission is denied for a component of the path prefix.

ELOOP Too many symbolic links were encountered in translating the pathname.

EINVAL The named file is not a symbolic link.

EIO An I/O error occurred while reading from the file system.

EFAULT *buf* extends outside the process's allocated address space.

ENOMEM Insufficient kernel memory was available.

CONFORMING TO

X/OPEN, 4.4BSD (the **readlink** function call appeared in 4.2BSD).

SEE ALSO

stat(2), **lstat**(2), **symlink**(2)

TOC

NAME

readv, writev - read or write a vector

SYNOPSIS

```
#include <sys/uio.h>

int readv(int fd, const struct iovec * vector, size_t count

int writev(int fd, const struct iovec * vector, size_t count

struct iovec {
  __ptr_t iov_base; /* Starting address.  */
  size_t iov_len; /* Length in bytes.  */
};
```

DESCRIPTION

readv reads data from file descriptor *fd*, and puts the result in the buffers described by *vector*. The number of buffers is specified by *count*. The buffers are filled in the order specified. Operates just like **read** except that data is put in *vector* instead of a contiguous buffer.

writev writes data to file descriptor *fd*, and from the buffers described by *vector*. The number of buffers is specified by *count*. The buffers are used in the order specified. Operates just like **write** except that data is taken from *vector* instead of a contiguous buffer.

RETURN VALUE

On success **readv** returns the number of bytes read. On success **writev** returns the number of bytes written. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL An invalid argument was given. For instance *count* might be greater than **MAX_IOVEC**, or zero. *fd* could also be attached to an object which is unsuitable for reading (for **readv**) or writing (for **writev**).

EFAULT "Segmentation fault." Most likely *vector* or some of the *iov_base* pointers points to memory that is not properly allocated.

EBADF The file descriptor *fd* is not valid.

EINTR The call was interrupted by a signal before any data was read/written.

EAGAIN Non-blocking I/O has been selected using **O_NONBLOCK** and no data was immediately available for reading. (Or the file descriptor *fd* is for an object that is locked.)

EISDIR *fd* refers to a directory.

EOPNOTSUP
fd refers to a socket or device that does not support reading/writing.

Other errors may occur, depending on the object connected to *fd*.

CONFORMING TO

4.4BSD (the **readv** and **writev** functions first appeared in BSD 4.2).

SEE ALSO

read(2), **write**(2), **fprintf**(3), **fscanf**(3)


TOC

TOC

NAME

reboot - reboot or enable/disable Ctrl-Alt-Del

SYNOPSIS

For libc4 and libc5 the library call and the system call are identical, and since kernel version 2.1.30 there are symbolic names LINUX_REBOOT_* for the constants and a fourth argument to the call:

#include <unistd.h>
#include <linux/reboot.h>

int reboot (int *magic*, **int** *magic2*, **int** *flag*

Under glibc some of the constants involved have gotten symbolic names RB_*, and the library call is a 1-argument wrapper around the 3-argument system call:

#include <unistd.h>
#include <sys/reboot.h>

int reboot (int *flag*));

DESCRIPTION

The **reboot** call reboots the system, or enables/disables the reboot keystroke (abbreviated CAD, since the default is Ctrl-Alt-Delete; it can be changed using **loadkeys**(1)).

This system call will fail (with EINVAL) unless *magic* equals LINUX_REBOOT_MAGIC1 (that is, 0xfee1dead) and *magic2* equals LINUX_REBOOT_MAGIC2 (that is, 672274793). However, since

2.1.17 also LINUX_REBOOT_MAGIC2A (that is, 85072278) and since 2.1.97 also LINUX_REBOOT_MAGIC2B (that is, 369367448) are permitted as value for *magic2*. (The hexadecimal values of these constants are meaningful.) The *flag* argument can have the following values:

LINUX_REBOOT_CMD_RESTART
(RB_AUTOBOOT, 0x1234567). The message `Restarting system.' is printed, and a default restart is performed immediately. If not preceded by a **sync**(2), data will be lost.

LINUX_REBOOT_CMD_HALT
(RB_HALT_SYSTEM, 0xcdef0123; since 1.1.76). The message `System halted.' is printed, and the system is halted. Control is given to the ROM monitor, if there is one. If not preceded by a **sync**(2), data will be lost.

LINUX_REBOOT_CMD_POWER_OFF
(0x4321fedc; since 2.1.30). The message `Power down.' is printed, the system is stopped, and all power is removed from the system, if possible. If not preceded by a **sync**(2), data will be lost.

LINUX_REBOOT_CMD_RESTART2
(0xa1b2c3d4; since 2.1.30). The message `Restarting system with command '%s'' is printed, and a restart (using the command string given in *arg)* is performed immediately. If not preceded by a **sync**(2), data will be lost.

LINUX_REBOOT_CMD_CAD_ON
(RB_ENABLE_CAD, 0x89abcdef). CAD is enabled. This means that the CAD keystroke will immediately cause the action associated to LINUX_REBOOT_CMD_RESTART.

LINUX_REBOOT_CMD_CAD_OFF
(RB_DISABLE_CAD, 0). CAD is disabled. This means that the CAD keystroke will cause a SIGINT signal to be sent to init (process 1), whereupon this process may decide upon a proper action (maybe: kill all processes, sync, reboot).

Only the super-user may use this function.

The precise effect of the above actions depends on the architecture. For the i386 architecture, the additional argument does not do anything at present (2.1.122), but the type of reboot can be determined by kernel command line arguments (`reboot=...') to be either warm or cold, and either hard or through the BIOS.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
Bad magic numbers or *flag*.

EPERM
A non-root user attempts to call **reboot**.

CONFORMING TO

reboot is Linux specific, and should not be used in programs intended to be portable.

SEE ALSO

sync(2), **bootparam**(7), **ctrlaltdel**(8), **halt**(8), **reboot**(8)

NAME

recv, recvfrom, recvmsg - receive a message from a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int recv(int s, void *buf, int len

int recvfrom(int s, void *buf, int len struct sockaddr
*from, int *fromlen));

int recvmsg(int s, struct msghdr *msg, unsigned int flags
```

DESCRIPTION

The **recvfrom** and **recvmsg** are used to receive messages from a socket, and may be used to receive data on a socket whether or not it is connection-oriented.

If *from* is non-nil, and the socket is not connection-oriented, the source address of the message is filled in. *Fromlen* is a value-result parameter, initialized to the size of the buffer associated with *from*, and modified on return to indicate the actual size of the address stored there.

The **recv** call is normally used only on a *connected* socket (see **connect**(2)) and is identical to **recvfrom** with a nil *from* parameter. As it is redundant, it may not be supported in future releases.

All three routines return the length of the message on successful completion. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on

the type of socket the message is received from (see **socket**(2)).

If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is non-blocking (see **fcntl**(2)) in which case the value -1 is returned and the external variable *errno* set to **EWOULDBLOCK**. The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested; this behavior is affected by the socket-level options **SO_RCVLOWAT** and **SO_RCVTIMEO** described in **getsockopt**(2).

The **select**(2) call may be used to determine when more data arrive.

The *flags* argument to a recv call is formed by *or*'ing one or more of the values:

The **MSG_OOB** flag requests receipt of out-of-band data that would not be received in the normal data stream. Some protocols place expedited data at the head of the normal data queue, and thus this flag cannot be used with such protocols. The **MSG_PEEK** flag causes the receive operation to return data from the beginning of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same data. The **MSG_WAITALL** flag requests that the operation block until the full request is satisfied. However, the call may still return less data than requested if a signal is caught, an error or disconnect occurs, or the next data to be received is of a different type than that returned.

The **recvmsg** call uses a *msghdr* structure to minimize the number of directly supplied parameters. This structure has the following form, as defined in *sys/socket.h*:

```
struct msghdr {
     caddr_t   msg_name; /* optional address */
     u_int     msg_namelen;   /* size of address */
     struct    iovec *msg_iov;     /* scatter/gather array */
     u_int     msg_iovlen;    /* # elements in msg_iov */
     caddr_t   msg_control;   /* ancillary data, see below */
     u_int     msg_controllen; /* ancillary data buffer len */
     int  msg_flags;     /* flags on received message */
};
```

Here *msg_name* and *msg_namelen* specify the destination address if the socket is unconnected; *msg_name* may be given as a null pointer if no names are desired or required. *Msg_iov* and *msg_iovlen* describe scatter gather locations, as discussed in **readv**(2). *Msg_control*, which has length *msg_controllen*, points to a buffer for other protocol control related messages or other miscellaneous ancillary data. The messages are of the form:

```
struct cmsghdr {
     u_int     cmsg_len; /* data byte count, including hdr */
     int  cmsg_level;    /* originating protocol */
     int  cmsg_type;     /* protocol-specific type */
/* followed by
     u_char    cmsg_data[]; */
};
```

As an example, one could use this to learn of changes in the data-stream in XNS/SPP, or in ISO, to obtain user-connection-request data by requesting a recvmsg with no data buffer provided immediately after an **accept** call.
Open file descriptors are now passed as ancillary data for **AF_UNIX** domain sockets, with *cmsg_level* set to **SOL_SOCKET** and *cmsg_type* set to **SCM_RIGHTS**.

The *msg_flags* field is set on return according to the message received. **MSG_EOR** indicates end-of-record; the data returned completed a record (generally used with sockets of type **SOCK_SEQPACKET**). **MSG_TRUNC** indicates that the trailing portion of a datagram was discarded because the datagram was larger than the buffer supplied. **MSG_CTRUNC** indicates that some control data were discarded due to lack of space in the buffer for ancillary data. **MSG_OOB** is returned to indicate that expedited or out-of-band data were received.

RETURN VALUES

These calls return the number of bytes received, or -1 if an error occurred.

ERRORS

EBADF The argument **s** is an invalid descriptor.

ENOTCONN
The socket is associated with a connection-oriented protocol and has not been connected (see **connect**(2) and **accept**(2)).

ENOTSOCK
The argument **s** does not refer to a socket.

EWOULDBLOCK
The socket is marked non-blocking, and the receive operation would block, or a receive timeout had been set, and the timeout expired before data were received.

EINTR The receive was interrupted by delivery of a signal before any data were available.

EFAULT The receive buffer pointer(s) point outside the process's address space.

CONFORMING TO

4.4BSD (these function calls first appeared in 4.2BSD).

SEE ALSO

fcntl(2), **read**(2), **select**(2),

NAME

rename - change the name or location of a file

SYNOPSIS

```
#include <stdio.h>

int rename(const char *oldpath, const char *newpath));
```

DESCRIPTION

rename renames a file, moving it between directories if required.

Any other hard links to the file (as created using **link**) are unaffected.

If *newpath* already exists it will be atomically replaced (subject to a few conditions - see ERRORS below), so that there is no point at which another process attempting to access *newpath* will find it missing.

If *newpath* exists but the operation fails for some reason or the system crashes **rename** guarantees to leave an instance of *newpath* in place.

However, when overwriting there will probably be a window in which both *oldpath* and *newpath* refer to the file being renamed.

If *oldpath* refers to a symbolic link the link is renamed; if *newpath* refers to a symbolic link the link will be overwritten.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EISDIR *newpath* is an existing directory, but *oldpath* is not a directory.

EXDEV *oldpath* and *newpath* are not on the same filesystem.

ENOTEMPTY
newpath is a non-empty directory.

EBUSY *newpath* exists and is the current working directory or root directory of some process.

EEXIST The new pathname contained a path prefix of the old.

EINVAL An attempt was made to make a directory a subdirectory of itself.

EMLINK *oldpath* already has the maximum number of links to it, or it was a directory and the directory containing *newpath* has the maximum number of links.

ENOTDIR A component used as a directory in *oldpath* or *newpath* is not, in fact, a directory.

EFAULT *oldpath* or *newpath* points outside your accessible address space.

EACCES Write access to the directory containing *oldpath* or *newpath* is not allowed for the process's effective uid, or one of the directories in *oldpath* or *newpath* did not allow search (execute) permission, or *old-*

path was a directory and did not allow write permission (needed to update the .. entry).

EPERM The directory containing *oldpath* has the sticky bit set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it, or the filesystem containing *pathname* does not support renaming of the type requested.

ENAMETOOLONG
oldpath or *newpath* was too long.

ENOENT A directory component in *oldpath* or *newpath* does not exist or is a dangling symbolic link.

ENOMEM Insufficient kernel memory was available.

EROFS The file is on a read-only filesystem.

ELOOP Too many symbolic links were encountered in resolving *oldpath* or *newpath*.

ENOSPC The device containing the file has no room for the new directory entry.

CONFORMING TO

POSIX, 4.3BSD, ANSI C

BUGS

On NFS filesystems, you can not assume that if the operation failed the file was not renamed. If the server does the rename operation and then crashes, the retransmitted RPC which will be processed when the server is up again causes a failure. The application is expected to deal with this. See *link*(2) for a similar problem.

SEE ALSO

link(2), **unlink**(2), **symlink**(2), **mv**(1)

TOC

TOC

NAME

rmdir - delete a directory

SYNOPSIS

```
#include <unistd.h>

int rmdir(const char *pathname));
```

DESCRIPTION

rmdir deletes a directory, which must be empty.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM The filesystem containing *pathname* does not support the removal of directories.

EFAULT *pathname* points outside your accessible address space.

EACCES Write access to the directory containing *pathname* was not allowed for the process's effective uid, or one of the directories in *pathname* did not allow search (execute) permission.

EPERM The directory containing *pathname* has the sticky-bit (**S_ISVTX**) set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it.

ENAMETOOLONG
pathname was too long.

ENOENT A directory component in *pathname* does not exist or is a dangling symbolic link.

ENOTDIR *pathname*, or a component used as a directory in *pathname*, is not, in fact, a directory.

ENOTEMPTY
pathname contains entries other than . and .. .

EBUSY *pathname* is the current working directory or root directory of some process.

ENOMEM Insufficient kernel memory was available.

EROFS *pathname* refers to a file on a read-only filesystem.

ELOOP Too many symbolic links were encountered in resolving *pathname*.

CONFORMING TO

SVr4, SVID, POSIX, BSD 4.3

BUGS

Infelicities in the protocol underlying NFS can cause the unexpected disappearance of directories which are still being used.

SEE ALSO

rename(2), **mkdir**(2), **chdir**(2), **rmdir**(1), **rm**(1)

TOC

NAME

sched_get_priority_max, sched_get_priority_min - get static priority range

SYNOPSIS

```
#include <sched.h>

int sched_get_priority_max(int policy));

int sched_get_priority_min(int policy));
```

DESCRIPTION

sched_get_priority_max returns the maximum priority value that can be used with the scheduling algorithm identified by *policy*. **sched_get_priority_min** returns the minimum priority value that can be used with the scheduling algorithm identified by *policy*. Supported *policy* values are *SCHED_FIFO*, *SCHED_RR*, and *SCHED_OTHER*.

Processes with numerically higher priority values are scheduled before processes with numerically lower priority values. Thus, the value returned by **sched_get_priority_max** will be greater than the value returned by **sched_get_priority_min**.

Linux allows the static priority value range 1 to 99 for *SCHED_FIFO* and *SCHED_RR* and the priority 0 for *SCHED_OTHER*. Scheduling priority ranges for the various policies are not alterable.

The range of scheduling priorities may vary on other POSIX

systems, thus it is a good idea for portable applications to use a virtual priority range and map it to the interval given by **sched_get_priority_max** and **sched_get_priority_min**. POSIX.1b requires a spread of at least 32 between the maximum and the minimum values for *SCHED_FIFO* and *SCHED_RR*.

POSIX systems on which **sched_get_priority_max** and **sched_get_priority_min** are available define *_POSIX_PRIORITY_SCHEDULING* in <unistd.h>.

RETURN VALUE

On success, **sched_get_priority_max** and **sched_get_priority_min** return the maximum/minimum priority value for the named scheduling policy. On error, -1 is returned, *errno* is set appropriately.

ERRORS

EINVAL The parameter *policy* does not identify a defined scheduling policy.

CONFORMING TO

POSIX.1b (formerly POSIX.4)

SEE ALSO

sched_setscheduler(2), **sched_getscheduler**(2), **sched_setparam**(2), **sched_getparam**(2).

sched_setscheduler(2) has a description of the Linux scheduling scheme.

Programming for the real world - by Bill O. Gallmeister, O'Reilly & Associates, Inc., ISBN 1-56592-074-0
IEEE Std 1003.1b-1993 (POSIX.1b standard)
*ISO/IEC 9945-***1**:*1996*

TOC

NAME

sched_rr_get_interval - get the SCHED_RR interval for the named process

SYNOPSIS

```
#include <sched.h>

int sched_rr_get_interval(pid_t pid, struct timespec *tp));

struct timespec {
    time_t  tv_sec;     /* seconds */
    long    tv_nsec;    /* nanoseconds */
};
```

DESCRIPTION

sched_rr_get_interval writes into the *timespec* structure pointed to by *tp* the round robin time quantum for the process identified by *pid*. If *pid* is zero, the time quantum for the calling process is written into **tp*. The identified process should be running under the *SCHED_RR* scheduling policy.

The round robin time quantum value is not alterable under Linux 1.3.81.

POSIX systems on which **sched_rr_get_interval** is available define *_POSIX_PRIORITY_SCHEDULING* in <unistd.h>.

RETURN VALUE

On success, **sched_rr_get_interval** returns 0. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

ESRCH The process whose ID is *pid* could not be found.

ENOSYS The system call is not yet implemented.

CONFORMING TO

POSIX.1b (formerly POSIX.4)

BUGS

As of Linux 1.3.81 **sched_rr_get_interval** returns with error ENOSYS, because SCHED_RR has not yet been fully implemented and tested properly.

SEE ALSO

sched_setscheduler(2) has a description of the Linux scheduling scheme.

Programming for the real world - by Bill O. Gallmeister, O'Reilly & Associates, Inc., ISBN 1-56592-074-0
IEEE Std 1003.1b-1993 (POSIX.1b standard, formerly POSIX.4)

TOC

TOC

NAME

sched_setparam, sched_getparam - set and get scheduling parameters

SYNOPSIS

```
#include <sched.h>

int sched_setparam(pid_t pid, const struct sched_param *p);

int sched_getparam(pid_t pid, struct sched_param *p);

struct sched_param {
    ...
    int sched_priority;
    ...
};
```

DESCRIPTION

sched_setparam sets the scheduling parameters associated with the scheduling policy for the process identified by *pid*. If *pid* is zero, then the parameters of the current process are set. The interpretation of the parameter **p** depends on the selected policy. Currently, the following three scheduling policies are supported under Linux: *SCHED_FIFO*, *SCHED_RR*, and *SCHED_OTHER*.

sched_getparam retrieves the scheduling parameters for the process identified by *pid*. If *pid* is zero, then the parameters of the current process are retrieved.

sched_setparam checks the validity of **p** for the scheduling

policy of the process. The parameter **p**->*sched_priority* must lie within the range given by **sched_get_priority_min** and **sched_get_priority_max**.

POSIX systems on which **sched_setparam** and **sched_getparam** are available define *_POSIX_PRIORITY_SCHEDULING* in <unistd.h>.

RETURN VALUE

On success, **sched_setparam** and **sched_getparam** return 0. On error, -1 is returned, *errno* is set appropriately.

ERRORS

ESRCH The process whose ID is *pid* could not be found.

EPERM The calling process does not have appropriate privileges. The process calling **sched_setparam** needs an effective uid equal to the euid or uid of the process identified by *pid*, or it must be a superuser process.

EINVAL The parameter **p** does not make sense for the current scheduling policy.

CONFORMING TO

POSIX.1b (formerly POSIX.4)

SEE ALSO

TOC

sched_setscheduler(2), **sched_getscheduler**(2), **sched_get_priority_max**(2), **sched_get_priority_min**(2), **nice**(2), **setpriority**(2), **getpriority**(2),

sched_setscheduler(2) has a description of the Linux scheduling scheme.

Programming for the real world - by Bill O. Gallmeister, O'Reilly & Associates, Inc., ISBN 1-56592-074-0
IEEE Std 1003.1b-1993 (POSIX.1b standard)
*ISO/IEC 9945-***1**:*1996*

TOC

TOC

NAME

sched_setscheduler, sched_getscheduler - set and get scheduling algorithm/parameters

SYNOPSIS

```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy, const struct
sched_param *p);

int sched_getscheduler(pid_t pid));

struct sched_param {
    ...
    int sched_priority;
    ...
};
```

DESCRIPTION

sched_setscheduler sets both the scheduling policy and the associated parameters for the process identified by *pid*. If *pid* equals zero, the scheduler of the calling process will be set. The interpretation of the parameter **p** depends on the selected policy. Currently, the following three scheduling policies are supported under Linux: *SCHED_FIFO*, *SCHED_RR*, and *SCHED_OTHER*; their respective semantics is described below.

sched_getscheduler queries the scheduling policy currently applied to the process identified by *pid*. If *pid* equals zero, the policy of the calling process will be retrieved.

Scheduling Policies

The scheduler is the kernel part that decides which runnable process will be executed by the CPU next. The Linux scheduler offers three different scheduling policies, one for normal processes and two for real-time applications. A static priority value *sched_priority* is assigned to each process and this value can be changed only via system calls. Conceptually, the scheduler maintains a list of runnable processes for each possible *sched_priority* value, and *sched_priority* can have a value in the range 0 to 99. In order to determine the process that runs next, the Linux scheduler looks for the non-empty list with the highest static priority and takes the process at the head of this list. The scheduling policy determines for each process, where it will be inserted into the list of processes with equal static priority and how it will move inside this list.

SCHED_OTHER is the default universal time-sharing scheduler policy used by most processes, *SCHED_FIFO* and *SCHED_RR* are intended for special time-critical applications that need precise control over the way in which runnable processes are selected for execution. Processes scheduled with *SCHED_OTHER* must be assigned the static priority 0, processes scheduled under *SCHED_FIFO* or *SCHED_RR* can have a static priority in the range 1 to 99. Only processes with superuser privileges can get a static priority higher than 0 and can therefore be scheduled under *SCHED_FIFO* or *SCHED_RR*. The system calls **sched_get_priority_min** and **sched_get_priority_max** can be used to to find out the valid priority range for a scheduling policy in a portable way on all POSIX.1b conforming systems.

All scheduling is preemptive: If a process with a higher static priority gets ready to run, the current process will be preempted and returned into its wait list. The scheduling policy only determines the ordering within the list of runnable processes with equal static priority.

SCHED_FIFO: First In-First out scheduling

SCHED_FIFO can only be used with static priorities higher than 0, that means that when a *SCHED_FIFO* processes becomes runnable, it will always preempt immediately any currently


TOC

running normal *SCHED_OTHER* process. *SCHED_FIFO* is a simple scheduling algorithm without time slicing. For processes scheduled under the *SCHED_FIFO* policy, the following rules are applied: A *SCHED_FIFO* process that has been preempted by another process of higher priority will stay at the head of the list for its priority and will resume execution as soon as all processes of higher priority are blocked again. When a *SCHED_FIFO* process becomes runnable, it will be inserted at the end of the list for its priority. A call to **sched_setscheduler** or **sched_setparam** will put the *SCHED_FIFO* process identified by *pid* at the end of the list if it was runnable. A process calling **sched_yield** will be put at the end of the list. No other events will move a process scheduled under the *SCHED_FIFO* policy in the wait list of runnable processes with equal static priority. A *SCHED_FIFO* process runs until either it is blocked by an I/O request, it is preempted by a higher priority process, or it calls **sched_yield**.

SCHED_RR: Round Robin scheduling

SCHED_RR is a simple enhancement of *SCHED_FIFO*. Everything described above for *SCHED_FIFO* also applies to *SCHED_RR*, except that each process is only allowed to run for a maximum time quantum. If a *SCHED_RR* process has been running for a time period equal to or longer than the time quantum, it will be put at the end of the list for its priority. A *SCHED_RR* process that has been preempted by a higher priority process and subsequently resumes execution as a running process will complete the unexpired portion of its round robin time quantum. The length of the time quantum can be retrieved by **sched_rr_get_interval**.

SCHED_OTHER: Default Linux time-sharing scheduling

SCHED_OTHER can only be used at static priority 0. *SCHED_OTHER* is the standard Linux time-sharing scheduler that is intended for all processes that do not require special static priority real-time mechanisms. The process to run is chosen from the static priority 0 list based on a dynamic priority that is determined only inside this list. The dynamic priority is based on the nice level (set by the **nice** or **setpriority** system call) and increased for each time quantum the process is ready to run, but denied to run by the scheduler. This ensures fair progress among all

SCHED_OTHER processes.

Response time

A blocked high priority process waiting for the I/O has a certain response time before it is scheduled again. The device driver writer can greatly reduce this response time by using a "slow interrupt" interrupt handler as described in **request_irq**(9).

Miscellaneous

Child processes inherit the scheduling algorithm and parameters across a **fork**.

Memory locking is usually needed for real-time processes to avoid paging delays, this can be done with **mlock** or **mlockall**.

As a non-blocking end-less loop in a process scheduled under *SCHED_FIFO* or *SCHED_RR* will block all processes with lower priority forever, a software developer should always keep available on the console a shell scheduled under a higher static priority than the tested application. This will allow an emergency kill of tested real-time applications that do not block or terminate as expected. As *SCHED_FIFO* and *SCHED_RR* processes can preempt other processes forever, only root processes are allowed to activate these policies under Linux.

POSIX systems on which **sched_setscheduler** and **sched_getscheduler** are available define *_POSIX_PRIORITY_SCHEDULING* in <unistd.h>.

RETURN VALUE

On success, **sched_setscheduler** returns zero. On success, **sched_getscheduler** returns the policy for the process (a non-negative integer). On error, -1 is returned, *errno* is set appropriately.

ERRORS

ESRCH The process whose ID is *pid* could not be found.

EPERM The calling process does not have appropriate privileges. Only root processes are allowed to activate the *SCHED_FIFO* and *SCHED_RR* policies. The process calling **sched_setscheduler** needs an effective uid equal to the euid or uid of the process identified by *pid*, or it must be a superuser process.

EINVAL The scheduling *policy* is not one of the recognized policies, or the parameter **p** does not make sense for the *policy*.

CONFORMING TO

POSIX.1b (formerly POSIX.4)

BUGS

As of linux-1.3.81, *SCHED_RR* has not yet been tested carefully and might not behave exactly as described or required by POSIX.1b.

SEE ALSO

sched_setparam(2), **sched_getparam**(2), **sched_yield**(2), **sched_get_priority_max**(2), **sched_get_priority_min**(2), **nice**(2), **setpriority**(2), **getpriority**(2), **mlockall**(2), **munlockall**(2), **mlock**(2), **munlock**(2).

Programming for the real world - by Bill O. Gallmeister, O'Reilly & Associates, Inc., ISBN 1-56592-074-0
IEEE Std 1003.1b-1993 (POSIX.1b standard)
*ISO/IEC 9945-***1**:*1996* - This is the new 1996 revision of POSIX.1 which contains in one single standard POSIX.1(1990), POSIX.1b(1993), POSIX.1c(1995), and POSIX.1i(1995).

TOC

NAME

sched_yield - yield the processor

SYNOPSIS

```
#include <sched.h>

int sched_yield(void);
```

DESCRIPTION

A process can relinquish the processor voluntarily without blocking by calling **sched_yield**. The process will then be moved to the end of the queue for its static priority and a new process gets to run.

Note: If the current process is the only process in the highest priority list at that time, this process will continue to run after a call to **sched_yield**.

POSIX systems on which **sched_yield** is available define *_POSIX_PRIORITY_SCHEDULING* in <unistd.h>.

RETURN VALUE

On success, **sched_yield** returns 0. On error, -1 is returned, and *errno* is set appropriately.

CONFORMING TO

POSIX.1b (formerly POSIX.4)

SEE ALSO

sched_setscheduler(2) for a description of Linux scheduling.

Programming for the real world - by Bill O. Gallmeister, O'Reilly & Associates, Inc., ISBN 1-56592-074-0
IEEE Std 1003.1b-1993 (POSIX.1b standard)
*ISO/IEC 9945-**1**:1996*

NAME

select, FD_CLR, FD_ISSET, FD_SET, FD_ZERO - synchronous I/O multiplexing

SYNOPSIS

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int n, fd_set *readfds, fd_set *writefds, fd_set
*exceptfds, struct timeval *timeout);

FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);
FD_SET(int fd, fd_set *set);
FD_ZERO(fd_set *set);
```

DESCRIPTION

select waits for a number of file descriptors to change status.

Three independent sets of descriptors are watched. Those listed in *readfds* will be watched to see if characters become available for reading, those in *writefds* will be watched to see if it is ok to immediately write on them, and those in *exceptfds* will be watched for exceptions. On exit, the sets are modified in place to indicate which descriptors actually changed status.

Four macros are provided to manipulate the sets. **FD_ZERO** will clear a set. **FD_SET** and **FD_CLR** add or remove a given

descriptor from a set. **FD_ISSET** tests to see if a descriptor is part of the set; this is useful after **select** returns.

n is the highest-numbered descriptor in any of the three sets, plus 1.

timeout is an upper bound on the amount of time elapsed before **select** returns. It may be zero, causing **select** to return immediately. If *timeout* is NULL (no timeout), **select** can block indefinitely.

RETURN VALUE

On success, **select** returns the number of descriptors contained in the descriptor sets, which may be zero if the timeout expires before anything interesting happens. On error, -1 is returned, and *errno* is set appropriately; the sets and *timeout* become undefined, so do not rely on their contents after an error.

ERRORS

EBADF An invalid file descriptor was given in one of the sets.

EINTR A non blocked signal was caught.

EINVAL **n** is negative.

ENOMEM **select** was unable to allocate memory for internal tables.


TOC

NOTES

Some code calls **select** with all three sets empty, **n** zero, and a non-null *timeout* as a fairly portable way to sleep with subsecond precision.

On Linux, *timeout* is modified to reflect the amount of time not slept; most other implementations do not do this. This causes problems both when Linux code which reads *timeout* is ported to other operating systems, and when code is ported to Linux that reuses a struct timeval for multiple **select**s in a loop without reinitializing it. Consider *timeout* to be undefined after **select** returns.

EXAMPLE

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int
main(void)
{
    fd_set rfds;
    struct timeval tv;
    int retval;

    /* Watch stdin (fd 0) to see when it has input. */
    FD_ZERO(&rfds);
    FD_SET(0, &rfds);
    /* Wait up to five seconds. */
    tv.tv_sec = 5;
    tv.tv_usec = 0;

    retval = select(1, &rfds, NULL, NULL, &tv);
    /* Don't rely on the value of tv now! */

    if (retval)
        printf("Data is available now.\n");
```


```
        /* FD_ISSET(0, &rfds) will be true. */
    else
        printf("No data within five seconds.\n");

    exit(0);

}
```

CONFORMING TO

4.4BSD (the **select** function first appeared in 4.2BSD). Generally portable to/from non-BSD systems supporting clones of the BSD socket layer (including System V variants). However, note that the System V variant typically sets the timeout variable before exit, but the BSD variant does not.

SEE ALSO

accept(2), **connect**(2), **read**(2), **recv**(2), **send**(2), **write**(2)

TOC

NAME

semctl - semaphore control operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
/* union semun is defined by including <sys/sem.h> */
#else
/* according to X/OPEN we have to define it ourselves */
union semun {
        int val;                    /* value for SETVAL */
        struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
        unsigned short int *array;  /* array for GETALL, SETALL */
        struct seminfo *__buf;      /* buffer for IPC_INFO */
};
#endif

int semctl (int semid, int semnum, int cmd, union semun arg))
```

DESCRIPTION

The function performs the control operation specified by *cmd* on the semaphore set (or on the *semnum*-th semaphore of the set) identified by *semid*. The first semaphore of the set is indicated by a value **0** for *semnum*.

Legal values for *cmd* are

IPC_STAT Copy info from the semaphore set data structure into the structure pointed to by *arg*.**buf**. The argument *semnum* is ignored. The calling process must have read access privileges on the sema-

phore set.

IPC_SET Write the values of some members of the **semid_ds** structure pointed to by *arg*.**buf** to the semaphore set data structure, updating also its **sem_ctime** member. Considered members from the user supplied **struct semid_ds** pointed to by *arg*.**buf** are

```
sem_perm.uid
sem_perm.gid
sem_perm.mode  /* only lowest 9-bits */
```

The calling process effective user-ID must be one among super-user, creator or owner of the semaphore set. The argument *semnum* is ignored.

IPC_RMID Remove immediately the semaphore set and its data structures awakening all waiting processes (with an error return and **errno** set to **EIDRM**). The calling process effective user-ID must be one among super-user, creator or owner of the semaphore set. The argument *semnum* is ignored.

GETALL Return **semval** for all semaphores of the set into *arg*.**array**. The argument *semnum* is ignored. The calling process must have read access privileges on the semaphore set.

GETNCNT The system call returns the value of **semncnt** for the *semnum*-th semaphore of the set (i.e. the number of processes waiting for an increase of **semval** for the *semnum*-th semaphore of the set). The calling process must have read access privileges on the semaphore set.

GETPID The system call returns the value of **sempid** for the *semnum*-th semaphore of the set (i.e. the pid of the process that executed the last **semop** call for the *semnum*-th semaphore of the set). The calling process must have read access privileges on the semaphore set.

GETVAL The system call returns the value of **semval** for the *semnum*-th semaphore of the set. The calling process must have read access privileges on the semaphore set.

GETZCNT The system call returns the value of **semzcnt** for

the *semnum*-th semaphore of the set (i.e. the number of processes waiting for **semval** of the *semnum*-th semaphore of the set to become 0). The calling process must have read access privileges on the semaphore set.

SETALL Set **semval** for all semaphores of the set using *arg*.**array**, updating also the **sem_ctime** member of the **semid_ds** structure associated to the set. Undo entries are cleared for altered semaphores in all processes. Processes sleeping on the wait queue are awakened if some **semval** becomes 0 or increases. The argument *semnum* is ignored. The calling process must have alter access privileges on the semaphore set.

SETVAL Set the value of **semval** to *arg*.**val** for the *semnum*-th semaphore of the set, updating also the **sem_ctime** member of the **semid_ds** structure associated to the set. Undo entry is cleared for altered semaphore in all processes. Processes sleeping on the wait queue are awakened if **semval** becomes 0 or increases. The calling process must have alter access privileges on the semaphore set.

RETURN VALUE

On fail the system call returns -**1** with **errno** indicating the error. Otherwise the system call returns a nonnegative value depending on *cmd* as follows:

GETNCNT the value of **semncnt**.

GETPID the value of **sempid**.

GETVAL the value of **semval**.

GETZCNT the value of **semzcnt**.


TOC

ERRORS

For a failing return, **errno** will be set to one among the following values:

EACCES The calling process has no access permissions needed to execute *cmd*.

EFAULT The address pointed to by *arg*.**buf** or *arg*.**array** isn't accessible.

EIDRM The semaphore set was removed.

EINVAL Invalid value for *cmd* or *semid*.

EPERM The argument *cmd* has value **IPC_SET** or **IPC_RMID** but the calling process effective user-ID has insufficient privileges to execute the command.

ERANGE The argument *cmd* has value **SETALL** or **SETVAL** and the value to which **semval** has to be set (for some semaphore of the set) is less than 0 or greater than the implementation value **SEMVMX**.

NOTES

The **IPC_INFO**, **SEM_STAT** and **SEM_INFO** control calls are used by the **ipcs**(8) program to provide information on allocated resources. In the future these can be modified as needed or moved to a proc file system interface.

The following system limit on semaphore sets affects a **semctl** call:

SEMVMX Maximum value for **semval**: implementation dependent (32767).

CONFORMING TO

SVr4, SVID. SVr4 documents more error conditions EINVAL and EOVERFLOW.

SEE ALSO

ipc(5), **shmget**(2), **shmat**(2), **shmdt**(2).

TOC

TOC

NAME

semget - get a semaphore set identifier

SYNOPSIS

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/sem.h>

int semget ( key_t key, int nsems, int semflg )
```

DESCRIPTION

The function returns the semaphore set identifier associated to the value of the argument *key*. A new set of *nsems* semaphores is created if *key* has value **IPC_PRIVATE** or *key* isn't **IPC_PRIVATE**, no existing message queue is associated to *key*, and **IPC_CREAT** is asserted in *semflg* (i.e. *semflg*&**IPC_CREAT** isn't zero). The presence in *semflg* of the fields **IPC_CREAT** and **IPC_EXCL** plays the same role, with respect to the existence of the semaphore set, as the presence of **O_CREAT** and **O_EXCL** in the mode argument of the **open**(2) system call: i.e. the **msgget** function fails if *semflg* asserts both **IPC_CREAT** and **IPC_EXCL** and a semaphore set already exists for *key*.

Upon creation, the lower 9 bits of the argument *semflg* define the access permissions (for owner, group and others) to the semaphore set in the same format, and with the same meaning, as for the access permissions parameter in the **open**(2) or **creat**(2) system calls (though the execute permissions are not used by the system, and write permissions, for a semaphore set, effectively means alter permissions).

Furthermore, while creating, the system call initializes the system semaphore set data structure **semid_ds** as follows:

sem_perm.cuid and **sem_perm.uid** are set to the effective user-ID of the calling process.

sem_perm.cgid and **sem_perm.gid** are set to the effective group-ID of the calling process.

The lowest order 9 bits of **sem_perm.mode** are set to the lowest order 9 bit of *semflg*.

sem_nsems is set to the value of *nsems*.

sem_otime is set to 0.

sem_ctime is set to the current time.

The argument *nsems* can be **0** (a don't care) when the system call isn't a create one. Otherwise *nsems* must be greater than **0** and less or equal to the maximum number of semaphores per semid, (**SEMMSL**).

If the semaphore set already exists, the access permissions are verified, and a check is made to see if it is marked for destruction.

RETURN VALUE

If successful, the return value will be the semaphore set identifier (a positive integer), otherwise **-1** with **errno** indicating the error.

ERRORS

For a failing return, **errno** will be set to one among the following values:

EACCES A semaphore set exists for *key*, but the calling process has no access permissions to the set.

EEXIST A semaphore set exists for **key** and *semflg* was asserting both **IPC_CREAT** and **IPC_EXCL**.

EIDRM The semaphore set is marked as to be deleted.

ENOENT No semaphore set exists for *key* and *semflg* wasn't asserting **IPC_CREAT**.

ENOMEM A semaphore set has to be created but the system has not enough memory for the new data structure.

ENOSPC A semaphore set has to be created but the system limit for the maximum number of semaphore sets (**SEMMNI**), or the system wide maximum number of semaphores (**SEMMNS**), would be exceeded.

NOTES

IPC_PRIVATE isn't a flag field but a **key_t** type. If this special value is used for *key*, the system call ignores everything but the lowest order 9 bits of *semflg* and creates a new semaphore set (on success).

The followings are limits on semaphore set resources affecting a **semget** call:

SEMMNI System wide maximum number of semaphore sets: policy dependent.

SEMMSL Maximum number of semaphores per semid: implementation dependent (500 currently).

SEMMNS System wide maximum number of semaphores: policy dependent. Values greater than **SEMMSL * SEMMNI** makes it irrelevant.

BUGS

Use of **IPC_PRIVATE** doesn't inhibit to other processes the access to the allocated semaphore set.

As for the files, there is currently no intrinsic way for a process to ensure exclusive access to a semaphore set. Asserting both **IPC_CREAT** and **IPC_EXCL** in *semflg* only ensures (on success) that a new semaphore set will be created, it doesn't imply exclusive access to the semaphore set.

The data structure associated with each semaphore in the set isn't initialized by the system call. In order to initialize those data structures, one has to execute a subsequent call to **semctl**(2) to perform a **SETVAL** or a **SETALL** command on the semaphore set.

CONFORMING TO

SVr4, SVID. SVr4 documents additional error conditions EINVAL, EFBIG, E2BIG, EAGAIN, ERANGE, EFAULT.

SEE ALSO

ftok(3), **ipc**(5), **semctl**(2), **semop**(2).


TOC

NAME

semop - semaphore operations

SYNOPSIS

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/sem.h>

int semop ( int semid, struct sembuf *sops, unsigned nsops )
```

DESCRIPTION

The function performs operations on selected members of the semaphore set indicated by *semid*. Each of the *nsops* elements in the array pointed to by *sops* specify an operation to be performed on a semaphore by a **struct sembuf** including the following members:

```
short sem_num;  /* semaphore number: 0 = first */
short sem_op;   /* semaphore operation */
short sem_flg;  /* operation flags */
```

Flags recognized in **sem_flg** are **IPC_NOWAIT** and **SEM_UNDO**. If an operation asserts **SEM_UNDO**, it will be undone when the process exits.

The system call semantic assures that the operations will be performed if and only if all of them will succeed. Each operation is performed on the **sem_num**-th semaphore of the semaphore set - where the first semaphore of the set is semaphore **0** - and is one among the following three.

If **sem_op** is a positive integer, the operation adds this value to **semval**. Furthermore, if **SEM_UNDO** is asserted for this operation, the system updates the process undo count for this semaphore. The operation always goes through, so no process sleeping can happen. The calling process must have alter permissions on the semaphore set.

If **sem_op** is zero, the process must have read access permissions on the semaphore set. If **semval** is zero, the operation goes through. Otherwise, if **IPC_NOWAIT** is asserted in **sem_flg**, the system call fails (undoing all previous actions performed) with **errno** set to **EAGAIN**. Otherwise **semzcnt** is incremented by one and the process sleeps until one of the following occurs:

- **semval** becomes 0, at which time the value of **semzcnt** is decremented.
- The semaphore set is removed: the system call fails with **errno** set to **EIDRM**.
- The calling process receives a signal that has to be caught: the value of **semzcnt** is decremented and the system call fails with **errno** set to **EINTR**.

If **sem_op** is less than zero, the process must have alter permissions on the semaphore set. If **semval** is greater than or equal to the absolute value of **sem_op**, the absolute value of **sem_op** is subtracted by **semval**. Furthermore, if **SEM_UNDO** is asserted for this operation, the system updates the process undo count for this semaphore. Then the operation goes through. Otherwise, if **IPC_NOWAIT** is asserted in **sem_flg**, the system call fails (undoing all previous actions performed) with **errno** set to **EAGAIN**. Otherwise **semncnt** is incremented by one and the process sleeps until one of the following occurs:

- **semval** becomes greater or equal to the absolute value of **sem_op**, at which time the value of **semncnt** is decremented, the absolute value of **sem_op** is subtracted from **semval** and, if **SEM_UNDO** is asserted for this operation, the system updates

the process undo count for this semaphore.

- The semaphore set is removed from the system: the system call fails with **errno** set to **EIDRM**.

- The calling process receives a signal that has to be caught: the value of **semncnt** is decremented and the system call fails with **errno** set to **EINTR**.

In case of success, the **sempid** member of the structure **sem** for each semaphore specified in the array pointed to by *sops* is set to the process-ID of the calling process. Furthermore both **sem_otime** and **sem_ctime** are set to the current time.

RETURN VALUE

If successful the system call returns **0**, otherwise it returns **-1** with **errno** indicating the error.

ERRORS

For a failing return, **errno** will be set to one among the following values:

E2BIG The argument *nsops* is greater than **SEMOPM**, the maximum number of operations allowed per system call.

EACCES The calling process has no access permissions on the semaphore set as required by one of the specified operations.

EAGAIN An operation could not go through and **IPC_NOWAIT** was asserted in its *sem_flg*.

EFAULT The address pointed to by *sops* isn't accessible.

EFBIG For some operation the value of **sem_num** is less than 0 or greater than or equal to the number of semaphores in the set.

EIDRM The semaphore set was removed.

EINTR Sleeping on a wait queue, the process received a signal that had to be caught.

EINVAL The semaphore set doesn't exist, or *semid* is less than zero, or *nsops* has a non-positive value.

ENOMEM The **sem_flg** of some operation asserted **SEM_UNDO** and the system has not enough memory to allocate the undo structure.

ERANGE For some operation **semop+semval** is greater than **SEMVMX**, the implementation dependent maximum value for **semval**.

NOTES

The **sem_undo** structures of a process aren't inherited by its child on execution of a **fork**(2) system call. They are instead inherited by the substituting process resulting by the execution of the **execve**(2) system call.

The followings are limits on semaphore set resources affecting a **semop** call:

SEMOPM Maximum number of operations allowed for one **semop** call: policy dependent.

SEMVMX Maximum allowable value for **semval**: implementation dependent (32767).

The implementation has no intrinsic limits for the adjust on exit maximum value (**SEMAEM**), the system wide maximum number of undo structures (**SEMMNU**) and the per process maximum number of undo entries system parameters.

TOC

BUGS

The system maintains a per process **sem_undo** structure for each semaphore altered by the process with undo requests. Those structures are free at process exit. One major cause for unhappiness with the undo mechanism is that it does not fit in with the notion of having an atomic set of operations an array of semaphores. The undo requests for an array and each semaphore therein may have been accumulated over many **semopt** calls. Should the process sleep when exiting, or should all undo operations be applied with the **IPC_NOWAIT** flag in effect? Currently those undo operations which go through immediately are applied, and those that require a wait are ignored silently. Thus harmless undo usage is guaranteed with private semaphores only.

CONFORMING TO

SVr4, SVID. SVr4 documents additional error conditions EIN-VAL, EFBIG, ENOSPC.

SEE ALSO

ipc(5), **semctl**(2), **semget**(2).


TOC

NAME

send, sendto, sendmsg - send a message from a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int send(int s, const void *msg, int len unsigned int flags));

int sendto(int s, const void *msg, int len unsigned int flags, const struct sockaddr *to, int tolen

int sendmsg(int s, const struct msghdr *msg, unsigned int flags));
```

DESCRIPTION

WARNING: This is a BSD man page. As of Linux 0.99.11, **sendmsg** was not implemented.

Send, **sendto**, and **sendmsg** are used to transmit a message to another socket. **Send** may be used only when the socket is in a *connected* state, while **sendto** and **sendmsg** may be used at any time.

The address of the target is given by *to* with *tolen* specifying its size. The length of the message is given by *len*. If the message is too long to pass atomically through the underlying protocol, the error **EMSGSIZE** is returned, and the message is not transmitted.

No indication of failure to deliver is implicit in a **send**. Locally detected errors are indicated by a return value of -1.

If no messages space is available at the socket to hold the message to be transmitted, then **send** normally blocks, unless

the socket has been placed in non-blocking I/O mode. The **select**(2) call may be used to determine when it is possible to send more data.

The *flags* parameter may include one or more of the following:

```
#define   MSG_OOB        0x1  /* process out-of-band data */
#define   MSG_DONTROUTE  0x4  /* bypass routing, use direct interface */
```

The flag **MSG_OOB** is used to send *out-of-band* data on sockets that support this notion (e.g. **SOCK_STREAM**); the underlying protocol must also support *out-of-band* data. **MSG_DONTROUTE** is usually used only by diagnostic or routing programs.
See **recv**(2) for a description of the *msghdr* structure.

RETURN VALUES

The call returns the number of characters sent, or -1 if an error occurred.

ERRORS

EBADF An invalid descriptor was specified.

ENOTSOCK
The argument **s** is not a socket.

EFAULT An invalid user space address was specified for a parameter.

EMSGSIZE
The socket requires that message be sent atomically, and the size of the message to be sent made this impossible.

EWOULDBLOCK
The socket is marked non-blocking and the requested operation would block.

ENOBUFS The system was unable to allocate an internal buffer. The operation may succeed when buffers become available.

ENOBUFS The output queue for a network interface was full. This generally indicates that the interface has

stopped sending, but may be caused by transient congestion.

CONFORMING TO

4.4BSD, SVr4 (these function calls appeared in 4.2BSD). The SVr4 versions docoment additional error conditions EINVAL, EINTR, EMSGSIZE, ENOSR, ENOMEM.

SEE ALSO

fcntl(2), **recv**(2), **select**(2), **socket**(2), **write**(2)

NAME

setfsgid - set group identity used for file system checks

SYNOPSIS

int setfsgid(uid_t *fsgid)*)

DESCRIPTION

setfsgid sets the group ID that the Linux kernel uses to check for all accesses to the file system. Normally, the value of *fsgid* will shadow the value of the effective group ID. In fact, whenever the effective group ID is changed, *fsgid* will also be changed to new value of effective group ID.

An explicit call to **setfsgid** is usually only used by programs such as the Linux NFS server that need to change what group ID is used for file access without a corresponding change in the real and effective group IDs. A change in the normal group IDs for a program such as the NFS server is a security hole that can expose it to unwanted signals from other group IDs.

setfsgid will only succeed if the caller is the superuser or if *fsgid* matches either the real group ID, effective group ID, saved group ID, or the current value of *fsgid*.

RETURN VALUE

On success, the previous value of *fsgid* is returned. On error, the current value of *fsgid* is returned.

CONFORMING TO

setfsgid is Linux specific and should not be used in programs intended to be portable.

BUGS

No error messages of any kind are returned to the caller. At the very least, **EPERM** should be returned when the call fails.

SEE ALSO

setfsuid(2)

TOC

NAME

setfsuid - set user identity used for file system checks

SYNOPSIS

int setfsuid(uid_t *fsuid)*)

DESCRIPTION

setfsuid sets the user ID that the Linux kernel uses to check for all accesses to the file system. Normally, the value of *fsuid* will shadow the value of the effective user ID. In fact, whenever the effective user ID is changed, *fsuid* will also be changed to new value of effective user ID.

An explict call to **setfsuid** is usually only used by programs such as the Linux NFS server that need to change what user ID is used for file access without a corresponding change in the real and effective user IDs. A change in the normal user IDs for a program such as the NFS server is a security hole that can expose it to unwanted signals from other user IDs.

setfsuid will only succeed if the caller is the superuser or if *fsuid* matches either the real user ID, effective user ID, saved user ID, or the current value of *fsuid*.

RETURN VALUE

On success, the previous value of *fsuid* is returned. On error, the current value of *fsuid* is returned.

CONFORMING TO

setfsuid is Linux specific and should not be used in programs intended to be portable.

BUGS

No error messages of any kind are returned to the caller. At the very least, **EPERM** should be returned when the call fails.

SEE ALSO

setfsgid(2)

NAME

setgid - set group identity

SYNOPSIS

#include <unistd.h>

int setgid(gid_t *gid)*)

DESCRIPTION

setgid sets the effective group ID of the current process. If the caller is the superuser, the real and saved group ID's are also set.

Under Linux, **setgid** is implemented like the POSIX version with the _POSIX_SAVED_IDS feature. This allows a setgid (other than root) program to drop all of its group privileges, do some un-privileged work, and then re-engage the original effective group ID in a secure manner.

If the user is root or the program is setgid root, special care must be taken. The **setgid** function checks the effective gid of the caller and if it is the superuser, all process related group ID's are set to *gid*. After this has occurred, it is impossible for the program to regain root privileges.

Thus, a setgid-root program wishing to temporarily drop root privileges, assume the identity of a non-root group, and then regain root privileges afterwards cannot use **setgid**. You can accomplish this with the (non-POSIX, BSD) call **setegid**.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM The user is not the super-user, and *gid* does not match the effective or saved group ID of the calling process.

CONFORMING TO

SVr4, SVID.

SEE ALSO

getgid(2), **setregid**(2), **setegid**(2)

NAME

setpgid, getpgid, setpgrp, getpgrp - set/get process group

SYNOPSIS

```
#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid));
pid_t getpgid(pid_t pid));
int setpgrp(void);
pid_t getpgrp(void);
```

DESCRIPTION

setpgid sets the process group ID of the process specified by *pid* to *pgid*. If *pid* is zero, the process ID of the current process is used. If *pgid* is zero, the process ID of the process specified by *pid* is used.

getpgid returns the process group ID of the process specified by *pid*. If *pid* is zero, the process ID of the current process is used.

In the Linux DLL 4.4.1 library, **setpgrp** simply calls **setpgid(0,0)**.

getpgrp is equivalent to **getpgid(0)**.

Process groups are used for distribution of signals, and by terminals to arbitrate requests for their input: processes that have the same process group as the terminal are foreground and may read, while others will block with a signal if they attempt to read.

These calls are thus used by programs such as **csh**(1) to create process groups in implementing job control. The **TIOCGPGRP** and **TIOCSPGRP** calls described in **termios**(4) are used to get/set the process group of the control terminal.

RETURN VALUE

On success, **setpgid** and **setpgrp** return zero. On error, -1 is returned, and *errno* is set appropriately.

getpgid returns a process group on success. On error, -1 is returned, and *errno* is set appropriately.

getpgrp always returns the current process group.

ERRORS

EINVAL *pgid* is less than 0.

EPERM Various permission violations.

ESRCH *pid* does not match any process.

CONFORMING TO

SVr4, POSIX, 4.4BSD.

CONFORMING TO

The functions **setpgid** and **getpgrp** conform to POSIX.1. The function **setpgrp** is from BSD 4.2. The function **getpgid** conforms to SVr4.

SEE ALSO

getuid(2), **setsid**(2), **tcsetpgrp**(3), **termios**(4)


TOC

NAME

setregid, setegid - set real and / or effective group ID

SYNOPSIS

```
#include <unistd.h>

int setregid(gid_t rgid, gid_t egid));
int setegid(gid_t egid));
```

DESCRIPTION

setregid sets real and effective group ID's of the current process. Un-privileged users may change the real group ID to the effective group ID and vice-versa.

Prior to Linux 1.1.38, the saved ID paradigm, when used with **setregid** or **setegid** was broken. Starting at 1.1.38, it is also possible to set the effective group ID from the saved group ID.

Only the super-user may make other changes.

Supplying a value of -1 for either the real or effective group ID forces the system to leave that ID unchanged.

Currently (libc-4.x.x), **setegid(***egid)*) is functionally equivalent to **setregid(-1,** *egid)*).

If the real group ID is changed or the effective group ID is set to a value not equal to the previous real group ID, the saved group ID will be set to the new effective group ID.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

BSD 4.3 (the **setregid** function call first appeared in 4.2BSD).

SEE ALSO

getgid(2), **setgid**(2)

NAME

setresuid, setresgid - set real, effective and saved user or group ID

SYNOPSIS

```
#include <unistd.h>

int setresuid(uid_t ruid, uid_t euid, uid_t suid
int setresgid(gid_t rgid, gid_t egid, gid_t sgid
```

DESCRIPTION

setresuid (introduced in Linux 2.1.44) sets the real, effective and saved user ID's of the current process.

Unprivileged user processes (i.e., processes with each of real, effective and saved user ID nonzero) may change the real, effective and saved user ID, each to one of: the current uid, the current effective uid or the current saved uid.

The super-user may set real, effective and saved user ID to arbitrary values.

If one of the parameters equals -1, the corresponding value is not changed.

Completely analogously, **setresgid** sets the real, effective and saved group ID's of the current process, with the same restrictions for processes with each of real, effective and saved user ID nonzero.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM
The current process was not privileged and tried to change the IDs is a not allowed way.

CONFORMING TO

This call is Linux-specific.

SEE ALSO

getuid(2), **setuid**(2), **getreuid**(2), **setreuid**(2), **getresuid**(2)

TOC

NAME

setreuid, seteuid - set real and / or effective user ID

SYNOPSIS

```
#include <unistd.h>

int setreuid(uid_t ruid, uid_t euid));
int seteuid(uid_t euid));
```

DESCRIPTION

setreuid sets real and effective user ID's of the current process. Un-privileged users may change the real user ID to the effective user ID and vice-versa.

Prior to Linux 1.1.37, the saved ID paradigm, when used with **setreuid** or **seteuid** was broken.

Starting at 1.1.37, it is also possible to set the effective user ID from the saved user ID.

Only the super-user may make other changes.

Supplying a value of -1 for either the real or effective user ID forces the system to leave that ID unchanged.

Currently **seteuid(***euid)*) is functionally equivalent to **setreuid(-1,** *euid)*).

If the real user ID is changed or the effective user ID is set to a value not equal to the previous real user ID, the saved user ID will be set to the new effective user ID.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM
The current process is not the super-user and changes other than (i) swapping the effective user ID with the real user ID or (ii) setting one to the value of the other or (iii) setting the effective user ID to the value of the saved user ID was specified.

CONFORMING TO

BSD 4.3 (the **setreuid** function call first appeared in 4.2BSD.)

SEE ALSO

getuid(2), **setuid**(2)

NAME

setsid - creates a session and sets the process group ID

SYNOPSIS

```
#include <unistd.h>

pid_t setsid(void);
```

DESCRIPTION

setsid() creates a new session if the calling process is not a process group leader. The calling process is the leader of the new session, the process group leader of the new process group, and has no controlling tty. The process group ID and session ID of the calling process are set to the PID of the calling process. The calling process will be the only process in this new process group and in this new session.

RETURN VALUE

The session ID of the calling process.

ERRORS

On error, -1 will be returned. The only error which can happen is EPERM. It is returned when the process group ID of any process equals the PID of the calling process. Thus, in particular, **setsid** fails if the calling process is already a process group leader.

NOTES

A process group leader is a process with process group ID equal to its PID. In order to be sure that **setsid** will succeed, fork and exit, and have the child do **setsid().**

CONFORMING TO

POSIX, SVr4.

SEE ALSO

setpgid(2), **setpgrp**(2)

TOC

NAME

setuid - set user identity

SYNOPSIS

#include <unistd.h>

int setuid(uid_t *uid)*)

DESCRIPTION

setuid sets the effective user ID of the current process. If the effective userid of the caller is root, the real and saved user ID's are also set.

Under Linux, **setuid** is implemented like the POSIX version with the _POSIX_SAVED_IDS feature. This allows a setuid (other than root) program to drop all of its user privileges, do some un-privileged work, and then re-engage the original effective user ID in a secure manner.

If the user is root or the program is setuid root, special care must be taken. The **setuid** function checks the effective uid of the caller and if it is the superuser, all process related user ID's are set to *uid*. After this has occurred, it is impossible for the program to regain root privileges.

Thus, a setuid-root program wishing to temporarily drop root privileges, assume the identity of a non-root user, and then regain root privileges afterwards cannot use **setuid**. You can accomplish this with the (non-POSIX, BSD) call **seteuid**.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM
The user is not the super-user, and *uid* does not match the effective or saved user ID of the calling process.

CONFORMING TO

SVr4, SVID, POSIX.1. Not quite compatible with the 4.4BSD call, which sets all of the real, saved, and effective user IDs. SVr4 documents an additional EINVAL error condition.

LINUX-SPECIFIC REMARKS

Linux has the concept of filesystem user ID, normally equal to the effective user ID. The **setuid** call also sets the filesystem user ID of the current process. See **setfsuid**(2).

If *uid* is different from the old effective uid, the process will be forbidden from leaving core dumps.

SEE ALSO

getuid(2), **setreuid**(2), **seteuid**(2), **setfsuid**(2)

TOC

TOC

NAME

setup - setup devices and file systems, mount root file system

SYNOPSIS

```
#include <unistd.h>

_syscall0(int, setup);

int setup(void);
```

DESCRIPTION

setup is called once from within *linux/init/main.c*. It calls initialization functions for devices and file systems configured into the kernel and then mounts the root file system.

No user process may call **setup**. Any user process, even a process with super-user permission, will receive **EPERM**.

RETURN VALUE

setup always returns -1 for a user process.

ERRORS

EPERM
Always, for a user process.

CONFORMING TO

This function is Linux specific, and should not be used in programs intended to be portable.

NAME

shmctl - shared memory control

SYNOPSIS

```
#include <sys/ipc.h>

#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf
```

DESCRIPTION

shmctl() allows the user to receive information on a shared memory segment, set the owner, group, and permissions of a shared memory segment, or destroy a segment. The information about the segment identified by *shmid* is returned in a *shmid_ds* structure:

```
struct shmid_ds {
     struct    ipc_perm shm_perm;  /* operation perms */
     int  shm_segsz;          /* size of segment (bytes) */
     time_t    shm_atime;          /* last attach time */
     time_t    shm_dtime;          /* last detach time */
     time_t    shm_ctime;          /* last change time */
     unsigned short shm_cpid; /* pid of creator */
     unsigned short shm_lpid; /* pid of last operator */
     short     shm_nattch;         /* no. of current attaches */
     /* the following are private */
     unsigned short   shm_npages;  /* size of segment (pages) */
     unsigned long   *shm_pages;
     struct shm_desc *attaches;    /* descriptors for attaches */
};
```

The fields in the member *shm_perm* can be set:


```
struct ipc_perm
{
  key_t  key;
  ushort uid;   /* owner euid and egid */
  ushort gid;
  ushort cuid;  /* creator euid and egid */
  ushort cgid;
  ushort mode;  /* lower 9 bits of access modes */
  ushort seq;   /* sequence number */
};
```

The following *cmds* are available:

IPC_STAT is used to copy the information about the shared memory segment into the buffer *buf*. The user must have **read** access to the shared memory segment.

IPC_SET is used to apply the changes the user has made to the *uid*, *gid*, or *mode* members of the *shm_perms* field. Only the lowest 9 bits of *mode* are used. The *shm_ctime* member is also updated. The user must be the owner, creator, or the super-user.

IPC_RMID is used to mark the segment as destroyed. It will actually be destroyed after the last detach. (I.e., when the *shm_nattch* member of the associated structure *shmid_ds* is zero.) The user must be the owner, creator, or the super-user.

The user *must* ensure that a segment is eventually destroyed; otherwise its pages that were faulted in will remain in memory or swap.

In addition, the **super-user** can prevent or allow swapping of a shared memory segment with the following *cmds*: (Linux only)

SHM_LOCK prevents swapping of a shared memory segment. The user must fault in any pages that are required to be present after locking is enabled.

SHM_UNLOCK allows the shared memory segment to be swapped out.

The **IPC_INFO**, **SHM_STAT** and **SHM_INFO** control calls are used

by the **ipcs**(8) program to provide information on allocated resources. In the future, these man be modified as needed or moved to a proc file system interface.

SYSTEM CALLS

fork()
After a **fork()** the child inherits the attached shared memory segments.

exec()
After an **exec()** all attached shared memory segments are detached (not destroyed).

exit()
Upon **exit()** all attached shared memory segments are detached (not destroyed).

RETURN VALUE

0 is returned on success, -1 on error.

ERRORS

On error, **errno** will be set to one of the following:

EACCES is returned if **IPC_STAT** is requested and *shm_perm.modes* does not allow read access for *msqid*.

EFAULT The argument *cmd* has value **IPC_SET** or **IPC_STAT** but the address pointed to by *buf* isn't accessible.

EINVAL is returned if *shmid* is not a valid identifier, or *cmd* is not a valid command.

EIDRM is returned if *shmid* points to a removed identifier.

EPERM is returned if **IPC_SET** or **IPC_RMID** is attempted, and the user is not the creator, the owner, or the super-user, and the user does not have permission granted to their group or to the world.

CONFORMING TO

SVr4, SVID. SVr4 documents additional error conditions EINVAL, ENOENT, ENOSPC, ENOMEM, EEXIST. Neither SVr4 nor SVID documents an EIDRM error condition.

SEE ALSO

shmget(2), **shmop**(2)

TOC

NAME

shmget - allocates a shared memory segment

SYNOPSIS

```
#include <sys/ipc.h>

#include <sys/shm.h>

int shmget(key_t key, int size, int shmflg
```

DESCRIPTION

shmget() returns the identifier of the shared memory segment associated to the value of the argument *key*. A new shared memory segment, with size equal to the round up of *size* to a multiple of **PAGE_SIZE**, is created if *key* has value **IPC_PRIVATE** or *key* isn't **IPC_PRIVATE**, no shared memory segment is associated to *key*, and **IPC_CREAT** is asserted in *shmflg* (i.e. *shmflg***&IPC_CREAT** isn't zero). The presence in

shmflg is composed of:

IPC_CREAT to create a new segment. If this flag is not used, then **shmget()** will find the segment associated with *key*, check to see if the user has permission to receive the *shmid* associated with the segment, and ensure the segment is not marked for destruction.

IPC_EXCL used with **IPC_CREAT** to ensure failure if the segment exists.

mode_flags (lowest 9 bits)
specifying the permissions granted to the owner,

group, and world. Presently, the execute permissions are not used by the system.

If a new segment is created, the access permissions from *shmflg* are copied into the *shm_perm* member of the *shmid_ds* structure that defines the segment. The *shmid_ds* structure:

```
struct shmid_ds {
     struct    ipc_perm shm_perm;  /* operation perms */
     int  shm_segsz;          /* size of segment (bytes) */
     time_t    shm_atime;          /* last attach time */
     time_t    shm_dtime;          /* last detach time */
     time_t    shm_ctime;          /* last change time */
     unsigned short shm_cpid; /* pid of creator */
     unsigned short shm_lpid; /* pid of last operator */
     short     shm_nattch;         /* no. of current attaches */
};


struct ipc_perm
{
  key_t  key;
  ushort uid;   /* owner euid and egid */
  ushort gid;
  ushort cuid;  /* creator euid and egid */
  ushort cgid;
  ushort mode;  /* lower 9 bits of shmflg */
  ushort seq;   /* sequence number */
};
```

Furthermore, while creating, the system call initializes the system shared memory segment data structure **shmid_ds** as follows:

shm_perm.cuid and **shm_perm.uid** are set to the effective user-ID of the calling process.

shm_perm.cgid and **shm_perm.gid** are set to the effective group-ID of the calling process.

The lowest order 9 bits of **shm_perm.mode** are set to the lowest order 9 bit of *shmflg*.

shm_segsz is set to the value of *size*.

shm_lpid, **shm_nattch**, **shm_atime** and **shm_dtime** are set to **0.**

shm_ctime is set to the current time.

If the shared memory segment already exists, the access permissions are verified, and a check is made to see if it is marked for destruction.

SYSTEM CALLS

fork() After a **fork()** the child inherits the attached shared memory segments.

exec() After an **exec()** all attached shared memory segments are detached (not destroyed).

exit() Upon **exit()** all attached shared memory segments are detached (not destroyed).

RETURN VALUE

A valid segment identifier, *shmid*, is returned on success, -1 on error.

ERRORS

On failure, **errno** is set to one of the following:

EINVAL is returned if **SHMMIN** > *size* or *size* > **SHMMAX**, or *size* is greater than size of segment.

EEXIST is returned if **IPC_CREAT | IPC_EXCL** was specified and the segment exists.

EIDRM is returned if the segment is marked as destroyed, or was removed.

ENOSPC is returned if all possible shared memory id's have been taken or if allocating a segment of

the requested *size* would cause the system to exceed the system-wide limit on shared memory

ENOENT is returned if no segment exists for the given *key*, and **IPC_CREAT** was not specified.

EACCES is returned if the user does not have permission to access the shared memory segment.

ENOMEM is returned if no memory could be allocated for segment overhead.

NOTES

IPC_PRIVATE isn't a flag field but a **key_t** type. If this special value is used for *key*, the system call ignores everything but the lowest order 9 bits of *shmflg* and creates a new shared memory segment (on success).

The followings are limits on shared memory segment resources affecting a **shmget** call:

SHMALL System wide maximum of shared memory pages: policy dependent.

SHMMAX Maximum size in bytes for a shared memory segment: implementation dependent (currently 4M).

SHMMIN Minimum size in bytes for a shared memory segment: implementation dependent (currently 1 byte, though **PAGE_SIZE** is the effective minimum size).

SHMMNI System wide maximum number of shared memory segments: implementation dependent (currently 4096).

The implementation has no specific limits for the per process maximum number of shared memory segments (**SHMSEG**).

BUGS

Use of **IPC_PRIVATE** doesn't inhibit to other processes the access to the allocated shared memory segment.

As for the files, there is currently no intrinsic way for a process to ensure exclusive access to a shared memory segment. Asserting both **IPC_CREAT** and **IPC_EXCL** in *shmflg* only ensures (on success) that a new shared memory segment will be created, it doesn't imply exclusive access to the segment.

CONFORMING TO

SVr4, SVID. SVr4 documents an additional error condition EEXIST. Neither SVr4 nor SVID documents an EIDRM condition.

SEE ALSO

ftok(3), **ipc**(5), **shmctl**(2), **shmat**(2), **shmdt**(2).

TOC

NAME

shmop - shared memory operations

SYNOPSIS

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/shm.h>

char *shmat ( int shmid, char *shmaddr, int shmflg )

int shmdt ( char *shmaddr))
```

DESCRIPTION

The function **shmat** attaches the shared memory segment identified by **shmid** to the data segment of the calling process. The attaching address is specified by *shmaddr* with one of the following criteria:

If *shmaddr* is **0**, the system tries to find an unmapped region in the range 1 - 1.5G starting from the upper value and coming down from there.

If *shmaddr* isn't **0** and **SHM_RND** is asserted in *shmflg*, the attach occurs at address equal to the rounding down of *shmaddr* to a multiple of **SHMLBA**. Otherwise *shmaddr* must be a page aligned address at which the attach occurs.

If **SHM_RDONLY** is asserted in *shmflg*, the segment is attached for reading and the process must have read access permissions to the segment. Otherwise the segment is attached for

read and write and the process must have read and write access permissions to the segment. There is no notion of write-only shared memory segment.

The **brk** value of the calling process is not altered by the attach. The segment will automatically detached at process exit. The same segment may be attached as a read and as a read-write one, and more than once, in the process's address space.

On a successful **shmat** call the system updates the members of the structure **shmid_ds** associated to the shared memory segment as follows:

shm_atime is set to the current time.

shm_lpid is set to the process-ID of the calling process.

shm_nattch is incremented by one.

Note that the attach succeeds also if the shared memory segment is marked as to be deleted.

The function **shmdt** detaches from the calling process's data segment the shared memory segment located at the address specified by *shmaddr*. The detaching shared memory segment must be one among the currently attached ones (to the process's address space) with *shmaddr* equal to the value returned by the its attaching **shat** call.

On a successful **shmdt** call the system updates the members of the structure **shmid_ds** associated to the shared memory segment as follows:

shm_dtime is set to the current time.

shm_lpid is set to the process-ID of the calling process.

shm_nattch is decremented by one. If it becomes 0 and the segment is marked for deletion, the segment is deleted.

The occupied region in the user space of the calling process is unmapped.

SYSTEM CALLS

fork()
After a **fork()** the child inherits the attached shared memory segments.

exec()
After an **exec()** all attached shared memory segments are detached (not destroyed).

exit()
Upon **exit()** all attached shared memory segments are detached (not destroyed).

RETURN VALUE

On a failure both functions return **-1** with **errno** indicating the error, otherwise **shmat** returns the address of the attached shared memory segment, and **shmdt** returns **0**.

ERRORS

When **shmat** fails, at return **errno** will be set to one among the following values:

EACCES The calling process has no access permissions for the requested attach type.

EINVAL Invalid *shmid* value, unaligned (i.e., not page-aligned and **SHM_RND** was not specified) or invalid *shmaddr* value, or failing attach at **brk**.

ENOMEM Could not allocate memory for the descriptor or for the page tables.

The function **shmdt** can fails only if there is no shared memory segment attached at *shmaddr*, in such a case at return **errno** will be set to **EINVAL**.

NOTES

On executing a **fork**(2) system call, the child inherits all the attached shared memory segments.

The shared memory segments attached to a process executing an **execve**(2) system call will not be attached to the resulting process.

The following is a system parameter affecting a **shmat** system call:

SHMLBA Segment low boundary address multiple. Must be page aligned. For the current implementation the **SHMBLA** value is **PAGE_SIZE**.

The implementation has no intrinsic limit to the per process maximum number of shared memory segments (**SHMSEG**)

CONFORMING TO

SVr4, SVID. SVr4 documents an additional error condition EMFILE.

SEE ALSO

ipc(5), **shmctl**(2), **shmget**(2).

TOC

NAME

shutdown - shut down part of a full-duplex connection

SYNOPSIS

#include <sys/socket.h>

int shutdown(int s, int *how*));

DESCRIPTION

The *shutdown* call causes all or part of a full-duplex connection on the socket associated with **s** to be shut down. If *how* is 0, further receives will be disallowed. If *how* is 1, further sends will be disallowed. If *how* is 2, further sends and receives will be disallowed.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF **s** is not a valid descriptor.

ENOTSOCK
s is a file, not a socket.

ENOTCONN
The specified socket is not connected.

CONFORMING TO

4.4BSD (the **shutdown** function call first appeared in 4.2BSD).

SEE ALSO

connect(2), **socket**(2)

TOC

NAME

sigaction, sigprocmask, sigpending, sigsuspend - POSIX sig-nal handling functions.

SYNOPSIS

```
#include <signal.h>

int sigaction(int signum, const struct sigaction *act,
struct sigaction *oldact));

int sigprocmask(int how, const sigset_t *set, sigset_t *old-
set));

int sigpending(sigset_t *set));

int sigsuspend(const sigset_t *mask));
```

DESCRIPTION

The **sigaction** system call is used to change the action taken by a process on receipt of a specific signal.

signum specifies the signal and can be any valid signal except **SIGKILL** and **SIGSTOP**.

If *act* is non-null, the new action for signal *signum* is installed from *act*. If *oldact* is non-null, the previous action is saved in *oldact*.

The **sigaction** structure is defined as

```
struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

sa_handler specifies the action to be associated with *signum* and may be **SIG_DFL** for the default action, **SIG_IGN** to ignore this signal, or a pointer to a signal handling function.

sa_mask gives a mask of signals which should be blocked during execution of the signal handler. In addition, the signal which triggered the handler will be blocked, unless the **SA_NODEFER** or **SA_NOMASK** flags are used.

sa_flags specifies a set of flags which modify the behaviour of the signal handling process. It is formed by the bitwise OR of zero or more of the following:

SA_NOCLDSTOP
: If *signum* is **SIGCHLD**, do not receive notification when child processes stop (i.e., when child processes receive one of **SIGSTOP**, **SIGTSTP**, **SIGTTIN** or **SIGTTOU**).

SA_ONESHOT or **SA_RESETHAND**
: Restore the signal action to the default state once the signal handler has been called. (This is the default behavior of the **signal**(2) system call.)

SA_RESTART
: Provide behaviour compatible with BSD signal semantics by making certain system calls restartable across signals.

SA_NOMASK or **SA_NODEFER**
: Do not prevent the signal from being received from within its own signal handler.

The *sa_restorer* element is obsolete and should not be used.

The **sigprocmask** call is used to change the list of currently blocked signals. The behaviour of the call is dependent on

the value of *how*, as follows.

SIG_BLOCK
The set of blocked signals is the union of the current set and the *set* argument.

SIG_UNBLOCK
The signals in *set* are removed from the current set of blocked signals. It is legal to attempt to unblock a signal which is not blocked.

SIG_SETMASK
The set of blocked signals is set to the argument *set*.

If *oldset* is non-null, the previous value of the signal mask is stored in *oldset*.

The **sigpending** call allows the examination of pending signals (ones which have been raised while blocked). The signal mask of pending signals is stored in *set*.

The **sigsuspend** call temporarily replaces the signal mask for the process with that given by *mask* and then suspends the process until a signal is received.

RETURN VALUES

sigaction, **sigprocmask**, **sigpending** and **sigsuspend** return 0 on success and -1 on error.

ERRORS

EINVAL
An invalid signal was specified. This will also be generated if an attempt is made to change the action

for **SIGKILL** or **SIGSTOP**, which cannot be caught.

EFAULT
act, *oldact*, *set* or *oldset* point to memory which is not a valid part of the process address space.

EINTR
System call was interrupted.

NOTES

It is not possible to block **SIGKILL** or **SIGSTOP** with the sigprocmask call. Attempts to do so will be silently ignored.

According to POSIX, the behaviour of a process is undefined after it ignores a SIGFPE, SIGILL, or SIGSEGV signal that was not generated by the *kill*() or the *raise*() functions. Integer division by zero has undefined result. On some architectures it will generate a SIGFPE signal. (Also dividing the most negative integer by -1 may generate SIGFPE.) Ignoring this signal might lead to an endless loop.

POSIX (B.3.3.1.3) disallows setting the action for SIGCHLD to SIG_IGN. The BSD and SYSV behaviours differ, causing BSD software that sets the action for SIGCHLD to SIG_IGN to fail on Linux.

The POSIX spec only defines **SA_NOCLDSTOP**. Use of other *sa_flags* is non-portable.

The **SA_RESETHAND** flag is compatible with the SVr4 flag of the same name.

The **SA_NODEFER** flag is compatible with the SVr4 flag of the same name under kernels 1.3.9 and newer. On older kernels the Linux implementation allowed the receipt of any signal, not just the one we are installing (effectively overriding any *sa_mask* settings).

The **SA_RESETHAND** and **SA_NODEFER** names for SVr4 compatibility


TOC

are present only in library versions 3.0.9 and greater.

sigaction can be called with a null second argument to query the current signal handler. It can also be used to check whether a given signal is valid for the current machine by calling it with null second and third arguments.

See **sigsetops**(3) for details on manipulating signal sets.

CONFORMING TO

POSIX, SVr4. SVr4 does not document the EINTR condition.

SEE ALSO

kill(1), **kill**(2), **killpg**(2), **siginterrupt**(3), **signal**(2), **signal**(7), **sigvec**(2)

TOC

TOC

NAME

sigblock, siggetmask, sigsetmask, sigmask - manipulate the signal mask

SYNOPSIS

```
#include <signal.h>

int sigblock(int mask));

int siggetmask(void);

int sigsetmask(int mask));

int sigmask(int signum));
```

DESCRIPTION

This interface is made obsolete by **sigprocmask**(2).

The **sigblock** system call adds the signals specified in *mask* to the set of signals currently being blocked from delivery.

The **sigsetmask** system call replaces the set of blocked signals totally with a new set specified in *mask*. Signals are blocked if the corresponding bit in *mask* is a 1.

The current set of blocked signals can be obtained using **siggetmask**.

The **sigmask** macro is provided to construct the mask for a given *signum*.

RETURN VALUES

siggetmask returns the current set of masked signals.

sigsetmask and **sigblock** return the previous set of masked signals.

NOTES

Prototypes for these functions are only available if **__USE_BSD** is defined before **<signal.h>** is included.

It is not possible to block **SIGKILL** or **SIGSTOP** - this restriction is silently imposed by the system.

CONFORMING TO

4.4BSD. These function calls appeared in BSD 4.3 and are deprecated. Use the POSIX signal facilities for new programs.

SEE ALSO

kill(2), **sigprocmask**(2), **signal**(7)

TOC

NAME

signal - ANSI C signal handling

SYNOPSIS

#include <signal.h>

void (*signal(int *signum*, **void (***handler*))(int)))(int);

DESCRIPTION

The **signal** system call installs a new signal handler for the signal with number *signum*. The signal handler is set to *handler* which may be a user specified function, or one of the following:

SIG_IGN
Ignore the signal.

SIG_DFL
Reset the signal to its default behavior.

The integer argument that is handed over to the signal handler routine is the signal number. This makes it possible to use one signal handler for several signals.

RETURN VALUE

signal returns the previous value of the signal handler, or **SIG_ERR** on error.

NOTES

Signal handlers cannot be set for **SIGKILL** or **SIGSTOP**.

Unlike on BSD systems, signals under Linux are reset to their default behavior when raised. However, if you include **<bsd/signal.h>** instead of **<signal.h>** then **signal** is redefined as **__bsd_signal** and signal has the BSD semantics. Both versions of **signal** are library routines built on top of **sigaction**(2).

If you're confused by the prototype at the top of this manpage, it may help to see it separated out thus:

typedef void (*sighandler_t)(int);
sighandler_t signal(int *signum*, **sighandler_t** *handler*));

According to POSIX, the behaviour of a process is undefined after it ignores a SIGFPE, SIGILL, or SIGSEGV signal that was not generated by the *kill*() or the *raise*() functions. Integer division by zero has undefined result. On some architectures it will generate a SIGFPE signal. (Also dividing the most negative integer by -1 may generate SIGFPE.) Ignoring this signal might lead to an endless loop.

According to POSIX (B.3.3.1.3) you must not set the action for SIGCHLD to SIG_IGN. Here the BSD and SYSV behaviours differ, causing BSD software that sets the action for SIGCHLD to SIG_IGN to fail on Linux.

CONFORMING TO

ANSI C

SEE ALSO

kill(1), **kill**(2), **killpg**(2), **sigaction**(2), **signal**(7), **sigsetops**(3), **alarm**(2).

TOC

NAME

sigpause - atomically release blocked signals and wait for interrupt

SYNOPSIS

#include <signal.h>

int sigpause(int *sigmask)*);

DESCRIPTION

This interface is made obsolete by **sigsuspend**(2).

sigpause assigns *sigmask* to the set of masked signals and then waits for a signal to arrive; on return the set of masked signals is restored.

sigmask is usually 0 to indicate that no signals are to be blocked. **sigpause** always terminates by being interrupted, returning -1 with *errno* set to **EINTR**.

CONFORMING TO

4.4BSD. The **sigpause** function call appeared in 4.3BSD and is deprecated.

SEE ALSO

sigsuspend(2), **kill**(2), **sigaction**(2), **sigblock**(2), **sigvec**(2)

TOC

NAME

sigreturn - return from signal handler and cleanup stack frame

SYNOPSIS

int sigreturn(unsigned long *__unused*));

DESCRIPTION

When the Linux kernel creates the stack frame for a signal handler, a call to **sigreturn** is inserted into the stack frame so that the the signal handler will call **sigreturn** upon return. This inserted call to **sigreturn** cleans up the stack so that the process can restart from where it was interrupted by the signal.

RETURN VALUE

sigreturn never returns.

WARNING

The **sigreturn** call is used by the kernel to implement signal handlers. It should **never** be called directly. Better yet,

the specific use of the *__unused* argument varies depending on the architecture.

CONFORMING TO

sigreturn is specific to Linux and should not be used in programs intended to be portable.

FILES

/usr/src/linux/arch/i386/kernel/signal.c
/usr/src/linux/arch/alpha/kernel/entry.S

SEE ALSO

kill(2), **signal**(2), **signal**(7)

NAME

sigvec - BSD software signal facilities

SYNOPSIS

#include <bsd/signal.h>

int sigvec(int *sig*, **struct sigvec** ***vec**, **struct sigvec** ***ovec**

DESCRIPTION

This interface is made obsolete by **sigaction**(2).

Under Linux **sigvec** is #define'd to **sigaction**, and provides at best a rough approximation of the BSD sigvec interface.

CONFORMING TO

BSD, SVr4

SEE ALSO

sigaction(2), **signal**(2)

TOC

NAME

socket - create an endpoint for communication

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol
```

DESCRIPTION

Socket creates an endpoint for communication and returns a descriptor.

The *domain* parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. These families are defined in the include file *sys/socket.h*. The currently understood formats are

```
AF_UNIX  (UNIX internal protocols)

AF_INET  (ARPA Internet protocols)

AF_ISO   (ISO protocols)

AF_NS    (Xerox Network Systems protocols)

AF_IMPLINK
         (IMP "host at IMP" link layer)
```

The socket has the indicated *type*, which specifies the semantics of communication. Currently defined types are:

```
SOCK_STREAM
SOCK_DGRAM
SOCK_RAW
SOCK_SEQPACKET
SOCK_RDM
```

A **SOCK_STREAM** type provides sequenced, reliable, two-way connection based byte streams. An out-of-band data transmission mechanism may be supported. A **SOCK_DGRAM** socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A **SOCK_SEQPACKET** socket may provide a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer may be required to read an entire packet with each read system call. This facility is protocol specific, and presently implemented only for **AF_NS**. **SOCK_RAW** sockets provide access to internal network protocols and interfaces. The types **SOCK_RAW**, which is available only to the super-user, and **SOCK_RDM**, which is planned, but not yet implemented, are not described here.

The *protocol* specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place; see **protocols**(5).

Sockets of type **SOCK_STREAM** are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a **connect**(2) call. Once connected, data may be transferred using **read**(2) and **write**(2) calls or some variant of the **send**(2) and **recv**(2) calls. When a session has been completed a **close**(2) may be performed. Out-of-band data may also be transmitted as described in **send**(2) and received as described in **recv**(2).

The communications protocols used to implement a **SOCK_STREAM** insure that data is not lost or duplicated. If a piece of

data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with -1 returns and with **ETIMEDOUT** as the specific code in the global variable *errno*. The protocols optionally keep sockets *warm* by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for a extended period (e.g. 5 minutes). A **SIGPIPE** signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

SOCK_SEQPACKET sockets employ the same system calls as **SOCK_STREAM** sockets. The only difference is that **read**(2) calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.

SOCK_DGRAM and **SOCK_RAW** sockets allow sending of datagrams to correspondents named in **send**(2) calls. Datagrams are generally received with **recvfrom**(2), which returns the next datagram with its return address.

An **fcntl**(2) call can be used to specify a process group to receive a **SIGURG** signal when the out-of-band data arrives. It may also enable non-blocking I/O and asynchronous notification of I/O events via **SIGIO**.
The operation of sockets is controlled by socket level *options*. These options are defined in the file *sys/socket.h*. **Setsockopt**(2) and **getsockopt**(2) are used to set and get options, respectively.

RETURN VALUES

A -1 is returned if an error occurs, otherwise the return value is a descriptor referencing the socket.

ERRORS

EPROTONOSUPPORT
The protocol type or the specified protocol is not supported within this domain.

EMFILE The per-process descriptor table is full.

ENFILE The system file table is full.

EACCES Permission to create a socket of the specified type and/or protocol is denied.

ENOBUFS Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.

CONFORMING TO

4.4BSD (the **socket** function call appeared in 4.2BSD). Generally portable to/from non-BSD systems supporting clones of the BSD socket layer (including System V variants).

SEE ALSO

accept(2), **bind**(2), **connect**(2), **getsockname**(2), **getsockopt**(2), **ioctl**(2), **read**(2), **recv**(2), **select**(2), **socketpair**(2), **write**(2)

"An Introductory 4.3 BSD Interprocess Communication Tutorial" is reprinted in *UNIX Programmer*'**s** *Supplementary Documents Volume* **1**

"BSD Interprocess Communication Tutorial" is reprinted in *UNIX Programmer*'**s** *Supplementary Documents Volume* **1**

TOC

NAME

socketcall - socket system calls

SYNOPSIS

int socketcall(int *call*, **unsigned long** ***args)**;

DESCRIPTION

socketcall is a common kernel entry point for the socket system calls. *call* determines which socket function to invoke. *args* points to a block containing the actual arguments, which are passed through to the appropriate call.

User programs should call the appropriate functions by their usual names. Only standard library implementors and kernel hackers need to know about **socketcall**.

CONFORMING TO

This call is specific to Linux, and should not be used in programs intended to be portable.

SEE ALSO

TOC

accept(2), **bind**(2), **connect**(2), **getpeername**(2), **getsockname**(2), **getsockopt**(2), **listen**(2), **recv**(2), **recvfrom**(2), **send**(2), **sendto**(2), **setsockopt**(2), **shutdown**(2), **socket**(2), **socketpair**(2)

NAME

socketpair - create a pair of connected sockets

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int socketpair(int d, int type, int protocol
```

DESCRIPTION

The call creates an unnamed pair of connected sockets in the specified domain **d**, of the specified *type*, and using the optionally specified *protocol*. The descriptors used in referencing the new sockets are returned in *sv*[0] and *sv*[1]. The two sockets are indistinguishable.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EMFILE Too many descriptors are in use by this process.

EAFNOSUPPORT
The specified address family is not supported on this machine.

EPROTONOSUPPORT
The specified protocol is not supported on this machine.

EOPNOSUPPORT
The specified protocol does not support creation of socket pairs.

EFAULT The address *sv* does not specify a valid part of the process address space.

CONFORMING TO

4.4BSD (the **socketpair** function call appeared in 4.2BSD). Generally portable to/from non-BSD systems supporting clones of the BSD socket layer (including System V variants).

SEE ALSO

read(2), **write**(2), **pipe**(2)

TOC

NAME

stat, fstat, lstat - get file status

SYNOPSIS

```
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *file_name, struct stat *buf));
int fstat(int filedes, struct stat *buf));
int lstat(const char *file_name, struct stat *buf));
```

DESCRIPTION

These functions return information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file.

stat stats the file pointed to by *file_name* and fills in *buf*.

lstat is identical to **stat**, only the link itself is stated, not the file that is obtained by tracing the links.

fstat is identical to stat, only the open file pointed to by *filedes* (as returned by *open*(2)) is stated in place of *file_name*.

They all return a *stat* structure, which contains the following fields:

```
struct stat
{
    dev_t         st_dev;      /* device */
```


```
        ino_t         st_ino;      /* inode */
        mode_t        st_mode;     /* protection */
        nlink_t       st_nlink;    /* number of hard links */
        uid_t         st_uid;      /* user ID of owner */
        gid_t         st_gid;      /* group ID of owner */
        dev_t         st_rdev;     /* device type (if inode device) */
        off_t         st_size;     /* total size, in bytes */
        unsigned long st_blksize;  /* blocksize for filesystem I/O */
        unsigned long st_blocks;   /* number of blocks allocated */
        time_t        st_atime;    /* time of last access */
        time_t        st_mtime;    /* time of last modification */
        time_t        st_ctime;    /* time of last change */
    };
```

Note that *st_blocks* may not always be in terms of blocks of size *st_blksize*, and that *st_blksize* may instead provide a notion of the "preferred" blocksize for efficient file system I/O.
Not all of the Linux filesystems implement all of the time fields. Traditionally, *st_atime* is changed by **mknod**(2), **utime**(2), **read**(2),

Traditionally, *st_mtime* is changed by **mknod**(2), **utime**(2), and **write**(2). The *st_mtime* is *not* changed for changes in owner, group, hard link count, or mode.

Traditionally, *st_ctime* is changed by writing or by setting inode information (i.e., owner, group, link count, mode, etc.).

The following POSIX macros are defined to check the file type:

S_ISLNK(m) is it a symbolic link?

S_ISREG(m) regular file?

S_ISDIR(m) directory?

S_ISCHR(m) character device?

S_ISBLK(m) block device?

S_ISFIFO(m) fifo?

S_ISSOCK(m) socket?

The following flags are defined for the *st_mode* field:

S_IFMT 00170000 bitmask for the file type bitfields (not POSIX)

S_IFSOCK 0140000 socket (not POSIX)

S_IFLNK 0120000 symbolic link (not POSIX)

S_IFREG 0100000 regular file (not POSIX)

S_IFBLK 0060000 block device (not POSIX)

S_IFDIR 0040000 directory (not POSIX)

S_IFCHR 0020000 character device (not POSIX)

S_IFIFO 0010000 fifo (not POSIX)

S_ISUID 0004000 set UID bit

S_ISGID 0002000 set GID bit

S_ISVTX 0001000 sticky bit (not POSIX)

S_IRWXU 00700 user (file owner) has read, write and execute permission

S_IRUSR 00400 user has read permission (same as S_IREAD, which is not POSIX)

S_IWUSR 00200 user has write permission (same as S_IWRITE, which is not POSIX)

S_IXUSR 00100 user has execute permission (same as S_IEXEC, which is not POSIX)

S_IRWXG 00070 group has read, write and execute permission

S_IRGRP 00040 group has read permission

S_IWGRP 00020 group has write permission

S_IXGRP 00010 group has execute permission

S_IRWXO 00007 others have read, write and execute permission

S_IROTH 00004 others have read permission

S_IWOTH 00002 others have write permisson

S_IXOTH 00001 others have execute permission

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF
filedes is bad.

ENOENT
A component of the path *file_name* does not exist, or the path is an empty string.

ENOTDIR
A component of the path is not a directory.

ELOOP
Too many symbolic links encountered while traversing the path.

EFAULT
Bad address.

EACCES
Permission denied.

ENOMEM
Out of memory (i.e. kernel memory).

ENAMETOOLONG
File name too long.

CONFORMING TO

The **stat** and **fstat** calls conform to SVr4, SVID, POSIX, X/OPEN, BSD 4.3. The **lstat** call conforms to 4.3BSD and SVr4. SVr4 documents additional **fstat** error conditions EINTR, ENOLINK, and EOVERFLOW. SVr4 documents additional **stat** and **lstat** error conditions EACCES, EINTR, EMULTIHOP, ENOLINK, and EOVERFLOW.

SEE ALSO

chmod(2), **chown**(2), **readlink**(2), **utime (2)**

NAME

statfs, fstatfs - get file system statistics

SYNOPSIS

```
#include <sys/vfs.h>

int statfs(const char *path, struct statfs *buf));
int fstatfs(int fd, struct statfs *buf));
```

DESCRIPTION

statfs returns information about a mounted file system. *path* is the path name of any file within the mounted filesystem. *buf* is a pointer to a *statfs* structure defined as follows:

```
struct statfs {
   long    f_type;     /* type of filesystem (see below) */
   long    f_bsize;    /* optimal transfer block size */
   long    f_blocks;   /* total data blocks in file system */
   long    f_bfree;    /* free blocks in fs */
   long    f_bavail;   /* free blocks avail to non-superuser */
   long    f_files;    /* total file nodes in file system */
   long    f_ffree;    /* free file nodes in fs */
   fsid_t  f_fsid;     /* file system id */
   long    f_namelen;  /* maximum length of filenames */
   long    f_spare[6]; /* spare for later */
};

File system types:

linux/affs_fs.h:
   AFFS_SUPER_MAGIC      0xADFF
```


```
linux/ext_fs.h:
   EXT_SUPER_MAGIC       0x137D
linux/ext2_fs.h:
   EXT2_OLD_SUPER_MAGIC  0xEF51
   EXT2_SUPER_MAGIC      0xEF53
linux/hpfs_fs.h:
   HPFS_SUPER_MAGIC      0xF995E849
linux/iso_fs.h:
   ISOFS_SUPER_MAGIC     0x9660
linux/minix_fs.h:
   MINIX_SUPER_MAGIC     0x137F /* orig. minix */
   MINIX_SUPER_MAGIC2    0x138F /* 30 char minix */
   MINIX2_SUPER_MAGIC    0x2468 /* minix V2 */
   MINIX2_SUPER_MAGIC2   0x2478 /* minix V2, 30 char names */
linux/msdos_fs.h:
   MSDOS_SUPER_MAGIC     0x4d44
linux/ncp_fs.h:
   NCP_SUPER_MAGIC       0x564c
linux/nfs_fs.h:
   NFS_SUPER_MAGIC       0x6969

linux/proc_fs.h:
   PROC_SUPER_MAGIC      0x9fa0
linux/smb_fs.h:
   SMB_SUPER_MAGIC       0x517B
linux/sysv_fs.h:
   XENIX_SUPER_MAGIC     0x012FF7B4
   SYSV4_SUPER_MAGIC     0x012FF7B5
   SYSV2_SUPER_MAGIC     0x012FF7B6
   COH_SUPER_MAGIC       0x012FF7B7
linux/ufs_fs.h:
   UFS_MAGIC             0x00011954
linux/xia_fs.h:
   _XIAFS_SUPER_MAGIC    0x012FD16D
```

Fields that are undefined for a particular file system are set to -1. **fstatfs** returns the same information about an open file referenced by descriptor *fd*.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

For **statfs**:

ENOTDIR A component of the path prefix of *path* is not a directory.

ENAMETOOLONG
path is too long.

ENOENT The file referred to by *path* does not exist.

EACCES Search permission is denied for a component of the path prefix of *path*.

ELOOP Too many symbolic links were encountered in translating *path*.

EFAULT *Buf* or *path* points to an invalid address.

EIO An I/O error occurred while reading from or writing to the file system.

ENOMEM Insufficient kernel memory was available.

ENOSYS The filesystem *path* is on does not support **statfs**.

For **fstatfs**:

EBADF *fd* is not a valid open file descriptor.

EFAULT *buf* points to an invalid address.

EIO An I/O error occurred while reading from or writing to the file system.

ENOSYS The filesystem *fd* is open on does not support **statfs**.

CONFORMING TO

4.4BSD.

SEE ALSO

stat(2)

TOC

NAME

stime - set time

SYNOPSIS

```
#include <time.h>

int stime(time_t *t);
```

DESCRIPTION

stime sets the system's idea of the time and date. Time, pointed to by **t**, is measured in seconds from 00:00:00 GMT January 1, 1970. **stime()** may only be executed by the super user.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM The caller is not the super-user.

CONFORMING TO

SVr4, SVID, X/OPEN

SEE ALSO

date(1)

TOC

NAME

swapon, swapoff - start/stop swapping to file/device

SYNOPSIS

```
#include <unistd.h>
#include <asm/page.h> /* to find PAGE_SIZE
#include <sys/swap.h>

int swapon(const char *path, int swapflags));
int swapoff(const char *path));
```

DESCRIPTION

swapon sets the swap area to the file or block device specified by *path*. **swapoff** stops swapping to the file or block device specified by *path*.

swapon takes a *swapflags* argument. If *swapflags* has the *SWAP_FLAG_PREFER* bit turned on, the new swap area will have a higher priority than default. The priority is encoded as:

(prio << SWAP_FLAG_PRIO_SHIFT) & SWAP_FLAG_PRIO_MASK

These functions may only be used by the super-user.

PRIORITY

Each swap area has a priority, either high or low. The default priority is low. Within the low-priority areas, newer areas are even lower priority than older areas.

All priorities set with *swapflags* are high-priority, higher than default. They may have any non-negative value chosen by the caller. Higher numbers mean higher priority.

Swap pages are allocated from areas in priority order, highest priority first. For areas with different priorities, a higher-priority area is exhausted before using a lower-priority area. If two or more areas have the same priority, and it is the highest priority available, pages are allocated on a round-robin basis between them.

As of Linux 1.3.6, the kernel usually follows these rules, but there are exceptions.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

Many other errors can occur if *path* is not valid.

EPERM The user is not the super-user, or more than **MAX_SWAPFILES** (defined to be 8 in Linux 1.3.6) are in use.

EINVAL is returned if *path* exists, but is neither a regular path nor a block device.

TOC

ENOENT is returned if *path* does not exist.

ENOMEM is returned if there is insufficient memory to start swapping.

CONFORMING TO

These functions are Linux specific and should not be used in programs intended to be portable. The second `swapflags' argument was introduced in Linux 1.3.2.

NOTES

The partition or path must be prepared with **mkswap**(8).

SEE ALSO

mkswap(8), **swapon**(8), **swapoff**(8)


TOC

NAME

symlink - make a new name for a file

SYNOPSIS

#include <unistd.h>

int symlink(const char ***oldpath*****, const char** ***newpath*****));**

DESCRIPTION

symlink creates a symbolic link named *newpath* which contains the string *oldpath*.

Symbolic links are interpreted at run-time as if the contents of the link had been substituted into the path being followed to find a file or directory.

Symbolic links may contain .. path components, which (if used at the start of the link) refer to the parent directories of that in which the link resides.

A symbolic link (also known as a soft link) may point to an existing file or to a nonexistent one; the latter case is known as a dangling link.

The permissions of a symbolic link are irrelevant; the ownership is ignored when following the link, but is checked when removal or renaming of the link is requested and the link is in a directory with the sticky bit set.

If *newpath* exists it will *not* be overwritten.

TOC

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM The filesystem containing *newpath* does not support the creation of symbolic links.

EFAULT *oldpath* or *newpath* points outside your accessible address space.

EACCES Write access to the directory containing *newpath* is not allowed for the process's effective uid, or one of the directories in *newpath* did not allow search (execute) permission.

ENAMETOOLONG
oldpath or *newpath* was too long.

ENOENT A directory component in *newpath* does not exist or is a dangling symbolic link, or *oldpath* is the empty string.

ENOTDIR A component used as a directory in *newpath* is not, in fact, a directory.

ENOMEM Insufficient kernel memory was available.

EROFS *newpath* is on a read-only filesystem.

EEXIST *newpath* already exists.

ELOOP Too many symbolic links were encountered in resolving *newpath*.

ENOSPC The device containing the file has no room for the

new directory entry.

EIO An I/O error occurred.

NOTES

No checking of *oldpath* is done.

Deleting the name referred to by a symlink will actually delete the file (unless it also has other hard links). If this behaviour is not desired, use **link**.

CONFORMING TO

SVr4, SVID, POSIX, BSD 4.3. SVr4 documents additional error codes SVr4, SVID, BSD 4.3, X/OPEN. SVr4 documents additional error codes EDQUOT and ENOSYS. See **open**(2) re multiple files with the same name, and NFS.

SEE ALSO

readlink(2), **link**(2), **unlink**(2), **rename**(2), **open**(2), **lstat**(2), **ln**(1)

TOC

NAME

sync - commit buffer cache to disk.

SYNOPSIS

```
#include <unistd.h>

int sync(void);
```

DESCRIPTION

sync first commits inodes to buffers, and then buffers to disk.

RETURN VALUE

sync always returns 0.

CONFORMING TO

SVr4, SVID, X/OPEN, BSD 4.3

BUGS

According to the standard specification (e.g., SVID), **sync()** schedules the writes, but may return before the actual writing is done. However, since version 1.3.20 Linux does actually wait. (This still does not guarantee data integrity: modern disks have large caches.)

SEE ALSO

bdflush(2), **fsync**(2), **fdatasync**(2),

TOC

NAME

none - list of all system calls

SYNOPSIS

Linux 2.0 system calls.

DESCRIPTION

As of Linux 2.0.34, there are 164 system calls listed in */usr/include/asm/unistd.h*. This man page lists them (providing hyperlinks if you read this with a browser).

_llseek(2), _newselect(2), _sysctl(2), access(2), acct(2), adjtimex(2), afs_syscall, alarm(2), bdflush(2), break, brk(2), chdir(2), chmod(2), chown(2), chroot(2), clone(2), close(2), creat(2), create_module(2), delete_module(2), dup(2), dup2(2), execve(2), exit(2), fchdir(2), fchmod(2), fchown(2), fcntl(2), fdatasync(2), flock(2), fork(2), fstat(2), fstatfs(2), fsync(2), ftime, ftruncate(2), get_kernel_syms(2), getdents(2), getegid(2), geteuid(2), getgid(2), getgroups(2), getitimer(2), getpgid(2), getpgrp(2), getpid(2), getppid(2), getpriority(2), getrlimit(2), getrusage(2), getsid(2), gettimeofday(2), getuid(2), gtty, idle(2), init_module(2), ioctl(2), ioperm(2), iopl(2), ipc(2), kill(2), link(2), lock, lseek(2), lstat(2), mkdir(2), mknod(2), mlock(2), mlockall(2), mmap(2), modify_ldt(2), mount(2), mprotect(2), mpx, mremap(2), msync(2), munlock(2), munlockall(2), munmap(2), nanosleep(2), nice(2), oldfstat, oldlstat, oldolduname, oldstat, olduname, open(2), pause(2), personality(2), phys, pipe(2), prof, profil, ptrace(2), quotactl(2), read(2), readdir(2), readlink(2), readv(2), reboot(2), rename(2),

rmdir(2), sched_get_priority_max(2), sched_get_-priority_min(2), sched_getparam(2), sched_getscheduler(2), sched_rr_get_interval(2), sched_setparam(2), sched_set-scheduler(2), sched_yield(2), select(2), setdomainname(2), setfsgid(2), setfsuid(2), setgid(2), setgroups(2), set-hostname(2), setitimer(2), setpgid(2), setpriority(2), set-regid(2), setreuid(2), setrlimit(2), setsid(2), set-timeofday(2), setuid(2), setup(2), sgetmask(2), sigac-tion(2), signal(2), sigpending(2), sigprocmask(2), sigre-turn(2), sigsuspend(2), socketcall(2), ssetmask(2), stat(2), statfs(2), stime(2), stty, swapoff(2), swapon(2), sym-link(2), sync(2), sysfs(2), sysinfo(2), syslog(2), time(2), times(2), truncate(2), ulimit, umask(2), umount(2), uname(2), unlink(2), uselib(2), ustat(2), utime(2), vhangup(2), vm86(2), wait4(2), waitpid(2), write(2), wri-tev(2).

Of the above, 5 are obsolete, namely oldfstat, oldlstat, oldolduname, oldstat and olduname (see also obsolete(2)), and 11 are unimplemented, namely afs_syscall, break, ftime, gtty, lock, mpx, phys, prof, profil, stty and ulimit (see also unimplemented(2)). However, ftime(3), profil(3) and ulimit(3) exist as library routines. The slot for phys is in use since 2.1.116 for umount2; phys will never be imple-mented.

Roughly speaking, the code belonging to the system call with number __NR_xxx defined in */usr/include/asm/unistd.h* can be found in the kernel source in the routine *sys_xxx*(). (The dispatch table for i386 can be found in */usr/src/linux/arch/i386/kernel/entry.S*.) There are many exceptions, however, mostly because older system calls were superseded by newer ones, and this has been treated somewhat unsystematically. Below the details for Linux 2.0.34.

The defines __NR_oldstat and __NR_stat refer to the routines sys_stat() and sys_newstat(), and similarly for *fstat* and *lstat*. Similarly, the defines __NR_oldolduname, __NR_olduname and __NR_uname refer to the routines sys_olduname(), sys_uname() and sys_newuname(). Thus, __NR_stat and __NR_uname have always referred to the latest version of the system call, and the older ones are for back-ward compatibility.

It is different with *select* and *mmap*. These use five or more parameters, and caused problems the way parameter pass-

ing on the i386 used to be set up. Thus, while other architectures have sys_select() ans sys_mmap() corresponding to __NR_select and __NR_mmap, on i386 one finds old_select() and old_mmap() (routines that use a pointer to a parameter block) instead. These days passing five parameters is not a problem anymore, and there is a __NR__newselect (used by libc 6) that corresponds directly to sys_select().

Two other system call numbers, __NR__llseek and __NR__sysctl have an additional underscore absent in sys_llseek() and sys_sysctl().

Then there is __NR_readdir corresponding to old_readdir(), which will read at most one directory entry at a time, and is superseded by sys_getdents().

Finally, the system call 166, with entry point sys_vm86() does not have a symbolic number at all. This version supersedes sys_vm86old() with number __NR_vm86.

NAME

sysctl - read/write system parameters

SYNOPSIS

```
#include <unistd.h>

#include <linux/unistd.h>

#include <linux/sysctl.h>

_syscall1(int, _sysctl, struct __sysctl_args *, args);

int _sysctl(struct __sysctl_args *args));
```

DESCRIPTION

The **_sysctl** call reads and/or writes kernel parameters. For example, the hostname, or the maximum number of open files. The argument has the form

```
struct __sysctl_args {
        int *name;        /* integer vector describing variable */
        int nlen;         /* length of this vector */
        void *oldval;     /* 0 or address where to store old value */
        size_t *oldlenp;  /* available room for old value,
                             overwritten by actual size of old value */
        void *newval;     /* 0 or address of new value */
        size_t newlen;    /* size of new value */
};
```

This call does a search in a tree structure, possibly resembling a directory tree under **/proc/sys**, and if the requested item is found calls some appropriate routine to read or modify the value.

EXAMPLE


```
#include <linux/unistd.h>
#include <linux/types.h>
#include <linux/sysctl.h>

_syscall1(int, _sysctl, struct __sysctl_args *, args);
int sysctl(int *name, int nlen, void *oldval, size_t *oldlenp,
           void *newval, size_t newlen)
{
        struct __sysctl_args args={name,nlen,oldval,oldlenp,newval,newlen};
        return _sysctl(&args);
}

#define SIZE(x) sizeof(x)/sizeof(x[0])
#define OSNAMESZ 100

char osname[OSNAMESZ];
int osnamelth;
int name[] = { CTL_KERN, KERN_OSTYPE };

main(){
        osnamelth = SIZE(osname);
        if (sysctl(name, SIZE(name), osname, &osnamelth, 0, 0))
                perror("sysctl");
        else
                printf("This machine is running %*s\n", osnamelth, osname);
        return 0;
}
```

RETURN VALUES

Upon successful completion, **_sysctl** returns 0. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

ERRORS

ENOTDIR *name* was not found.

EPERM No search permission for one of the encountered `directories', or no read permission where *oldval*

was nonzero, or no write permission where *newval* was nonzero.

EFAULT The invocation asked for the previous value by setting *oldval* non-NULL, but allowed zero room in *oldlenp*.

CONFORMING TO

This call is Linux-specific, and should not be used in programs intended to be portable. A **sysctl** call has been present in Linux since version 1.3.57. It originated in 4.4BSD. Only Linux has the */proc/sys* mirror, and the object naming schemes differ between Linux and BSD 4.4, but the declaration of the **sysctl**(2) function is the same in both.

BUGS

The object names vary between kernel versions. THIS MAKES THIS SYSTEM CALL WORTHLESS FOR APPLICATIONS. Use the */proc/sys* interface instead.
Not all available objects are properly documented.
It is not yet possible to change operating system by writing to */proc/sys/kernel/ostype*.

SEE ALSO

proc(5).

NAME

sysfs - get file system type information

SYNOPSIS

int sysfs(int *option*, **const char *** *fsname)*);

int sysfs(int *option*, **unsigned int** *fs_index*, **char *** *buf*

int sysfs(int *option)*);

DESCRIPTION

sysfs returns information about the file system types currently present in the kernel. The specific form of the **sysfs** call and the information returned depends on the *option* in effect:

1 Translate the file-system identifier string *fsname* into a file-system type index.

2 Translate the file-system type index *fs_index* into a null-terminated file-system identifier string. This string will be written to the buffer pointed to by *buf*. Make sure that *buf* has enough space to accept the string.

3 Return the total number of file system types currently present in the kernel.

The numbering of the file-system type indexes begins with

zero.

RETURN VALUE

On success, **sysfs** returns the file-system index for option **1**, zero for option **2**, and the number of currently configured file systems for option **3**. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
fsname is not a valid file-system type identifier; *fs_index* is out-of-bounds; *option* is invalid.

EFAULT
Either *fsname* or *buf* is outside your accessible address space.

CONFORMING TO

SVr4.

NAME

sysinfo - returns information on overall system statistics

SYNOPSIS

```
#include <linux/kernel.h>
#include <linux/sys.h>

int sysinfo(struct sysinfo *info));
```

DESCRIPTION

sysinfo returns information in the following structure:

```
struct sysinfo {
     long uptime;              /* Seconds since boot */
     unsigned long loads[3];   /* 1, 5, and 15 minute load averages */
     unsigned long totalram;   /* Total usable main memory size */
     unsigned long freeram;    /* Available memory size */
     unsigned long sharedram;  /* Amount of shared memory */
     unsigned long bufferram;  /* Memory used by buffers */
     unsigned long totalswap;  /* Total swap space size */
     unsigned long freeswap;   /* swap space still available */
     unsigned short procs;     /* Number of current processes */
     char _f[22];              /* Pads structure to 64 bytes */
};
```

sysinfo provides a simple way of getting overall system statistics. This is more portable than reading */dev/kmem*. For an example of its use, see intro(2).

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EFAULT pointer to *struct sysinfo* is invalid

CONFORMING TO

This function is Linux-specific, and should not be used in programs intended to be portable.

The Linux kernel has a sysinfo system call since 0.98.pl6. Linux libc contains a sysinfo() routine since 5.3.5, and glibc has one since 1.90.

NAME

syslog - read and/or clear kernel message ring buffer; set console_loglevel

SYNOPSIS

#include <unistd.h>

#include <linux/unistd.h>

_syscall3(int, syslog, int, type, char *,

int syslog(int *type*, **char** ***bufp**, **int** *len*

DESCRIPTION

This is probably not the function you are interested in. Look at **syslog**(3) for the C library interface. This page only documents the bare kernel system call interface.

The *type* argument determines the action taken by **syslog**.

Quoting from *kernel/printk.c*:

```
/*
 * Commands to sys_syslog:
 *
 *      0 -- Close the log.  Currently a NOP.
 *      1 -- Open the log. Currently a NOP.
 *      2 -- Read from the log.
 *      3 -- Read up to the last 4k of messages in the ring buffer.
 *      4 -- Read and clear last 4k of messages in the ring buffer
 *      5 -- Clear ring buffer.
 *      6 -- Disable printk's to console
 *      7 -- Enable printk's to console
```


```
 *      8 -- Set level of messages printed to console
 */
```

Only function 3 is allowed to non-root processes.

The kernel log buffer
The kernel has a cyclic buffer of length LOG_BUF_LEN (4096, since 1.3.54: 8192, since 2.1.113: 16384) in which messages given as argument to the kernel function *printk*() are stored (regardless of their loglevel).

The call **syslog** (2,*buf*,*len*) waits until this kernel log buffer is nonempty, and then reads at most *len* bytes into the buffer *buf*. It returns the number of bytes read. Bytes read from the log disappear from the log buffer: the information can only be read once. This is the function executed by the kernel when a user program reads */proc/kmsg*.

The call **syslog** (3,*buf*,*len*) will read the last *len* bytes from the log buffer (nondestructively), but will not read more than was written into the buffer since the last `clear ring buffer' command (which does not clear the buffer at all). It returns the number of bytes read.

The call **syslog** (4,*buf*,*len*) does precisely the same, but also executes the `clear ring buffer' command.

The call **syslog** (5,*dummy*,*idummy*) only executes the `clear ring buffer' command.

The loglevel
The kernel routine *printk*() will only print a message on the console, if it has a loglevel less than the value of the variable *console_loglevel* (initially DEFAULT_CONSOLE_LOGLEVEL (7), but set to 10 if the kernel commandline contains the word `debug', and to 15 in case of a kernel fault - the 10 and 15 are just silly, and equivalent to 8). This variable is set (to a value in the range 1-8) by the call **syslog** (8,*dummy*,*value*). The calls **syslog** (*type*,*dummy*,*idummy* with *type* equal to 6 or 7, set it to 1 (kernel panics only) or 7 (all except debugging messages), respectively.

Every text line in a message has its own loglevel. This level is DEFAULT_MESSAGE_LOGLEVEL - 1 (6) unless the line starts with <d> where **d** is a digit in the range 1-7, in

which case the level is **d**. The conventional meaning of the loglevel is defined in <*linux/kernel.h*> as follows:

```
#define KERN_EMERG    "<0>"  /* system is unusable                */
#define KERN_ALERT    "<1>"  /* action must be taken immediately */
#define KERN_CRIT     "<2>"  /* critical conditions              */
#define KERN_ERR      "<3>"  /* error conditions                 */
#define KERN_WARNING  "<4>"  /* warning conditions               */
#define KERN_NOTICE   "<5>"  /* normal but significant condition */
#define KERN_INFO     "<6>"  /* informational                    */
#define KERN_DEBUG    "<7>"  /* debug-level messages             */
```

RETURN VALUE

In case of error, -1 is returned, and *errno* is set. Otherwise, for *type* equal to 2, 3 or 4, **syslog**() returns the number of bytes read, and otherwise 0.

ERRORS

EPERM
An attempt was made to change console_loglevel or clear the kernel message ring buffer by a process without root permissions.

EINVAL
Bad parameters.

ERESTARTSYS
System call was interrupted by a signal - nothing was read.

CONFORMING TO

This system call is Linux specific and should not be used in programs intended to be portable.

SEE ALSO

syslog(3)

TOC

TOC

NAME

times - get process times

SYNOPSIS

#include <sys/times.h>

clock_t times(struct tms *buf));

DESCRIPTION

times stores the current process times in *buf*.

struct tms is as defined in */usr/include/sys/times.h*:

```
struct  tms  {
              clock_t tms_utime;  /* user time */
              clock_t tms_stime;  /* system time */
              clock_t tms_cutime; /* user time of children */
              clock_t tms_cstime; /* system time of children */
              };
```

times returns the number of clock ticks that have elapsed since the system has been up.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, BSD 4.3

SEE ALSO

time(1), **getrusage**(2), **wait**(2)

NAME

truncate, ftruncate - truncate a file to a specified length

SYNOPSIS

```
#include <unistd.h>

int truncate(const char *path, size_t length));
int ftruncate(int fd, size_t length));
```

DESCRIPTION

Truncate causes the file named by *path* or referenced by *fd* to be truncated to at most *length* bytes in size. If the file previously was larger than this size, the extra data is lost. With **ftruncate**, the file must be open for writing.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

For **truncate**:

ENOTDIR A component of the path prefix is not a directory.

EINVAL The pathname contains a character with the high-order bit set.

ENAMETOOLONG
A component of a pathname exceeded 255 characters, or an entire path name exceeded 1023 characters.

ENOENT The named file does not exist.

EACCES Search permission is denied for a component of the path prefix.

EACCES The named file is not writable by the user.

ELOOP Too many symbolic links were encountered in translating the pathname.

EISDIR The named file is a directory.

EROFS The named file resides on a read-only file system.

ETXTBSY The file is a pure procedure (shared text) file that is being executed.

EIO An I/O error occurred updating the inode.

EFAULT *Path* points outside the process's allocated address space.

For **Ftruncate**:

EBADF The *fd* is not a valid descriptor.

EINVAL The *fd* references a socket, not a file.

EINVAL The *fd* is not open for writing.

CONFORMING TO

4.4BSD, SVr4 (these function calls first appeared in BSD 4.2). SVr4 documents additional **truncate** error conditions EINTR, EMFILE, EMULTIHP, ENAMETOOLONG, ENFILE, ENOLINK, ENOTDIR. SVr4 **ftruncate** documents additional EAGAIN and EINTR error conditions.

BUGS

These calls should be generalized to allow ranges of bytes in a file to be discarded.

SEE ALSO

open(2)

TOC

NAME

umask - set file creation mask

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

mode_t umask(mode_t mask);
```

DESCRIPTION

umask sets the umask to *mask* & 0777.

The umask is used by **open**(2) to set initial file permissions on a newly-created file. Specifically, permissions in the umask are turned off from the **mode** argument to **open**(2) (so, for example, the common umask default value of 022 results in new files being created with permissions 0666 & ~022 = 0644 = rw-r--r-- in the usual case where the **mode** is specified as 0666).

RETURN VALUE

This system call always succeeds and the previous value of the mask is returned.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, BSD 4.3

SEE ALSO

creat(2), **open**(2)

NAME

uname - get name and information about current kernel

SYNOPSIS

```
#include <sys/utsname.h>

int uname(struct utsname *buf));
```

DESCRIPTION

uname returns system information in *buf*. The *utsname* struct is as defined in <*sys/utsname.h*>:

```
struct utsname {
                char sysname[SYS_NMLN];
                char nodename[SYS_NMLN];
                char release[SYS_NMLN];
                char version[SYS_NMLN];
                char machine[SYS_NMLN];
                char domainname[SYS_NMLN];
                };
```

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EFAULT *buf* is not valid.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN

SEE ALSO

uname(1), **getdomainname**(2), **gethostname**(2)

NAME

none - undocumented system calls

SYNOPSIS

Undocumented system calls.

DESCRIPTION

As of Linux 2.0.34, there are 164 system calls listed in */usr/include/asm/unistd.h*. This man page mentions those calls which are implemented in the kernel but not yet documented in man pages. Some of these calls do not yet have prototypes in the libc include files.

SOLICITATION

If you have information about these system calls, please look in the kernel source code, write a man page (using a style similar to that of the other Linux section 2 man pages), and send it to **aeb@cwi.nl** for inclusion in the next man page release from the Linux Documentation Project.

STATUS

There are presently no undocumented system calls.

SEE ALSO

obsolete(2), **unimplemented**(2)

TOC

NAME

afs_syscall, break, ftime, gtty, lock, mpx, phys, prof, profil, stty, ulimit - unimplemented system calls

SYNOPSIS

Unimplemented system calls.

DESCRIPTION

These system calls are not implemented in the Linux 2.0 kernel.

RETURN VALUE

These system calls always return -1 and set *errno* to **ENOSYS**.

NOTES

Note that *ftime*(3), *profil*(3) and *ulimit*(3) are implemented as library functions.

Some system calls, like *ioperm*(2), *iopl*(2), *ptrace*(2) and *vm86*(2) only exist on certain architectures.

Some system calls, like *ipc*(2) and {*create*,*init*,*delete*}*_module*(2) only exist when the Linux kernel was built with support for them.

SEE ALSO

obsolete(2), **undocumented**(2).

NAME

unlink - delete a name and possibly the file it refers to

SYNOPSIS

```
#include <unistd.h>

int unlink(const char *pathname));
```

DESCRIPTION

unlink deletes a name from the filesystem. If that name was the last link to a file and no processes have the file open the file is deleted and the space it was using is made available for reuse.

If the name was the last link to a file but any processes still have the file open the file will remain in existence until the last file descriptor referring to it is closed.

If the name referred to a symbolic link the link is removed.

If the name referred to a socket, fifo or device the name for it is removed but processes which have the object open may continue to use it.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EFAULT *pathname* points outside your accessible address space.

EACCES Write access to the directory containing *pathname* is not allowed for the process's effective uid, or one of the directories in *pathname* did not allow search (execute) permission.

EPERM The directory containing *pathname* has the sticky-bit (**S_ISVTX**) set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it, or *pathname* is a directory.

ENAMETOOLONG
pathname was too long.

ENOENT A directory component in *pathname* does not exist or is a dangling symbolic link.

ENOTDIR A component used as a directory in *pathname* is not, in fact, a directory.

EISDIR *pathname* refers to a directory.

ENOMEM Insufficient kernel memory was available.

EROFS *pathname* refers to a file on a read-only filesystem.

ELOOP Too many symbolic links were encountered in translating *pathname*.

EIO An I/O error occurred.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, 4.3BSD. SVr4 documents additional error conditions EBUSY, EINTR, EMULTIHOP, ETXTBUSY, ENOLINK.

BUGS

Infelicities in the protocol underlying NFS can cause the unexpected disappearance of files which are still being used.

SEE ALSO

link(2), **rename**(2), **open**(2), **rmdir**(2), **mknod**(2), **mkfifo**(3), **remove**(3), **rm**(1)

TOC

NAME

uselib - select shared library

SYNOPSIS

#include <unistd.h>

int uselib(const char * *library)*);

DESCRIPTION

uselib selects the shared library binary that will be used by this processes.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

In addition to all of the error codes returned by **open**(2) and **mmap**(2), the following may also be returned:

ENOEXEC
The file specified by *library* is not executable, or does not have the correct magic numbers.

EACCES
The library specified by *library* is not readable.

CONFORMING TO

uselib() is Linux specific, and should not be used in programs intended to be portable.

SEE ALSO

open(2), **mmap**(2), **ldd**(1),

NAME

ustat - get file system statistics

SYNOPSIS

```
#include <sys/types.h>

int ustat(dev_t dev, struct ustat * ubuf));
```

DESCRIPTION

ustat returns information about a mounted file system. *dev* is a device number identifying a device containing a mounted file system. *ubuf* is a pointer to a ustat structure that contains the following members:

```
daddr_t f_tfree;              /* Total free blocks */
ino_t   f_tinode;             /* Number of free inodes */
char    f_fname[6];           /* Filsys name */
char    f_fpack[6];           /* Filsys pack name */
```

The last two fields, f_fname and f_fpack, are not implemented and will always be filled with null characters.

RETURN VALUE

On success, zero is returned and the ustat structure pointed to by *ubuf* will be filled in. On error, -1 is returned, and

errno is set appropriately.

ERRORS

EINVAL
dev does not refer to a device containing a mounted file system.

EFAULT
ubuf points outside of your accessible address space.

ENOSYS
The mounted file system referenced by *dev* does not support this operation, or any version of Linux before 1.3.16.

NOTES

ustat has only been provided for compatibility. All new programs should use **statfs**(2) instead.

CONFORMING TO

SVr4. SVr4 documents additional error conditions ENOLINK, ECOMM, and EINTR but has no ENOSYS condition.

SEE ALSO

statfs(2), **stat**(2)


TOC

TOC

TOC

NAME

utime, utimes - change access and/or modification times of an inode

SYNOPSIS

```
#include <sys/types.h>
#include <utime.h>

int utime(const char *filename, struct utimbuf *buf));


#include <sys/time.h>

int utimes(char *filename, struct timeval *tvp));
```

DESCRIPTION

utime changes the access and modification times of the inode specified by *filename* to the *actime* and *modtime* fields of *buf* respectively. If *buf* is **NULL**, then the access and modification times of the file are set to the current time. The *utimbuf* structure is:

```
struct utimbuf {
        time_t actime;  /* access time */
        time_t modtime; /* modification time */
};
```

In the Linux DLL 4.4.1 libraries, **utimes** is just a wrapper for **utime**: *tvp*[0].*tv_sec* is *actime*, and *tvp*[1].*tv_sec* is *modtime*. The *timeval* structure is:


```
struct timeval {
        long    tv_sec;         /* seconds */
        long    tv_usec;        /* microseconds */
};
```

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

Other errors may occur.

EACCES Permission to write the file is denied.

ENOENT *filename* does not exist.

CONFORMING TO

utime: SVr4, SVID, POSIX. SVr4 documents additional error conditions EFAULT, EINTR, ELOOP, EMULTIHOP, ENAMETOOLONG, ENOLINK, ENOTDIR, ENOLINK, ENOTDIR, EPERM, EROFS.
utimes: BSD 4.3

SEE ALSO

stat(2)


TOC

NAME

vhangup - virtually hangup the current tty

SYNOPSIS

```
#include <unistd.h>

int vhangup(void);
```

DESCRIPTION

vhangup simulates a hangup on the current terminal. This call arranges for other users to have a "clean" tty at login time.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM The user is not the super-user.

CONFORMING TO

This call is Linux-specific, and should not be used in programs intended to be portable.

SEE ALSO

init(8)

NAME

vm86old, vm86 - enter virtual 8086 mode

SYNOPSIS

```
#include <sys/vm86.h>

int vm86old(struct vm86_struct * info));

int vm86(unsigned long fn, struct vm86plus_struct * v86));
```

DESCRIPTION

The system call **vm86** was introduced in Linux 0.97p2. In Linux 2.1.15 and 2.0.28 it was renamed to **vm86old**, and a new **vm86** was introduced. The definition of `struct vm86_struct' was changed in 1.1.8 and 1.1.9.

These calls cause the process to enter VM86 mode, and are used by **dosemu**.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

(for **vm86old**)

EPERM Saved kernel stack exists. (This is a kernel sanity check; the saved stack should only exist within vm86 mode itself.)

CONFORMING TO

This call is specific to Linux on Intel processors, and should not be used in programs intended to be portable.

TOC

NAME

wait, waitpid - wait for process termination

SYNOPSIS

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status))
pid_t waitpid(pid_t pid, int *status, int options
```

DESCRIPTION

The **wait** function suspends execution of the current process until a child has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function. If a child has already exited by the time of the call (a so-called "zombie" process), the function returns immediately. Any system resources used by the child are freed.

The **waitpid** function suspends execution of the current process until a child as specified by the *pid* argument has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function. If a child as requested by *pid* has already exited by the time of the call (a so-called "zombie" process), the function returns immediately. Any system resources used by the child are freed.

The value of *pid* can be one of:

< -1 which means to wait for any child process whose process

group ID is equal to the absolute value of *pid*.

-1 which means to wait for any child process; this is the same behaviour which **wait** exhibits.

0 which means to wait for any child process whose process group ID is equal to that of the calling process.

> 0 which means to wait for the child whose process ID is equal to the value of *pid*.

The value of *options* is an OR of zero or more of the following constants:

WNOHANG which means to return immediately if no child has exited.

WUNTRACED
which means to also return for children which are stopped, and whose status has not been reported.

If *status* is not **NULL**, **wait** or **waitpid** store status information in the location pointed to by *status*.

This status can be evaluated with the following macros (these macros take the stat buffer (an **int**) as an argument - not a pointer to the buffer!):

WIFEXITED(*status)*)
is non-zero if the child exited normally.

WEXITSTATUS(*status)*)
evaluates to the least significant eight bits of the return code of the child which terminated, which may have been set as the argument to a call to **exit()** or as the argument for a **return** statement in the main program. This macro can only be evaluated if **WIFEXITED** returned non-zero.

WIFSIGNALED(*status)*)
returns true if the child process exited because of a signal which was not caught.

WTERMSIG(*status)*)
returns the number of the signal that caused the child process to terminate. This macro can only be

evaluated if **WIFSIGNALED** returned non-zero.

WIFSTOPPED(*status)*)
returns true if the child process which caused the return is currently stopped; this is only possible if the call was done using **WUNTRACED**.

WSTOPSIG(*status)*)
returns the number of the signal which caused the child to stop. This macro can only be evaluated if **WIFSTOPPED** returned non-zero.

RETURN VALUE

The process ID of the child which exited, -1 on error or zero if **WNOHANG** was used and no child was available (in which case, *errno* is set to an appropriate value).

ERRORS

ECHILD if the process specified in *pid* does not exist or is not a child of the calling process. (This can happen for one's own child if the action for SIGCHLD is set to SIG_IGN.)

EINVAL if the *options* argument was invalid.

ERESTARTSYS
if **WNOHANG** was not set and an unblocked signal or a **SIGCHLD** was caught. This error is returned by the system call. The library interface is not allowed to return **ERESTARTSYS**, but will return **EINTR**.

NOTES

The Single Unix Specification describes a flag SA_NOCLDWAIT (not present under Linux) such that if either this flag is set, or the action for SIGCHLD is set to SIG_IGN (which, by the way, is not allowed by POSIX), then children that exit do not become zombies and a call to *wait*() or *waitpid*() will block until all children have exited, and then fail with *errno* set to ECHILD.

CONFORMING TO

SVr4, POSIX.1

SEE ALSO

signal(2), **wait4**(2), **signal**(7)

TOC

NAME

wait3, wait4 - wait for process termination, BSD style

SYNOPSIS

```
#define _USE_BSD
#include <sys/types.h>
#include <sys/resource.h>
#include <sys/wait.h>


pid_t wait3(int *status, int options,
      struct rusage *rusage))

pid_t wait4(pid_t pid, int *status, int options
      struct rusage *rusage))
```

DESCRIPTION

The **wait3** function suspends execution of the current process until a child has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function. If a child has already exited by the time of the call (a so-called "zombie" process), the function returns immediately. Any system resources used by the child are freed.

The **wait4** function suspends execution of the current process until a child as specified by the *pid* argument has exited, or until a signal is delivered whose action is to terminate the current process or to call a signal handling function. If a child as requested by *pid* has already exited by the time of the call (a so-called "zombie" process), the func-

tion returns immediately. Any system resources used by the child are freed.

The value of *pid* can be one of:

< -1 which means to wait for any child process whose process group ID is equal to the absolute value of *pid*.

-1 which means to wait for any child process; this is equivalent to calling **wait3**.

0 which means to wait for any child process whose process group ID is equal to that of the calling process.

> 0 which means to wait for the child whose process ID is equal to the value of *pid*.

The value of *options* is a bitwise OR of zero or more of the following constants:

WNOHANG which means to return immediately if no child is there to be waited for.

WUNTRACED
which means to also return for children which are stopped, and whose status has not been reported.

If *status* is not **NULL**, **wait3** or **wait4** store status information in the location pointed to by *status*.

This status can be evaluated with the following macros (these macros take the stat buffer (an **int**) as an argument - not a pointer to the buffer!):

WIFEXITED(*status)*)
is non-zero if the child exited normally.

WEXITSTATUS(*status)*)
evaluates to the least significant eight bits of the return code of the child which terminated, which may have been set as the argument to a call to **exit()** or as the argument for a **return** statement in the main program. This macro can only be evaluated if **WIFEXITED** returned non-zero.

WIFSIGNALED(*status)*)

returns true if the child process exited because of a signal which was not caught.

WTERMSIG(*status)*)
returns the number of the signal that caused the child process to terminate. This macro can only be evaluated if **WIFSIGNALED** returned non-zero.

WIFSTOPPED(*status)*)
returns true if the child process which caused the return is currently stopped; this is only possible if the call was done using **WUNTRACED**.

WSTOPSIG(*status)*)
returns the number of the signal which caused the child to stop. This macro can only be evaluated if **WIFSTOPPED** returned non-zero.

If *rusage* is not **NULL**, the **struct rusage** as defined in *<sys/resource.h>* it points to will be filled with accounting information. See **getrusage**(2) for details.

RETURN VALUE

The process ID of the child which exited, -1 on error (in particular, when no unwaited-for child processes of the specified kind exist) or zero if **WNOHANG** was used and no child was available yet. In the latter two cases *errno* will be set appropriately.

ERRORS

ECHILD
No unwaited-for child process as specified does exist.

ERESTARTSYS
if **WNOHANG** was not set and an unblocked signal or a

SIGCHLD was caught. This error is returned by the system call. The library interface is not allowed to return **ERESTARTSYS**, but will return **EINTR**.

CONFORMING TO

SVr4, POSIX.1

SEE ALSO

signal(2), **getrusage**(2), **wait**(2),

TOC

NAME

write - write to a file descriptor

SYNOPSIS

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count
```

DESCRIPTION

write writes up to *count* bytes to the file referenced by the file descriptor *fd* from the buffer starting at *buf*. POSIX requires that a **read()** which can be proved to occur after a **write()** has returned returns the new data. Note that not all file systems are POSIX conforming.

RETURN VALUE

On success, the number of bytes written are returned (zero indicates nothing was written). On error, -1 is returned, and *errno* is set appropriately. If *count* is zero and the file descriptor refers to a regular file, 0 will be returned without causing any other effect. For a special file, the results are not portable.

ERRORS

EBADF
fd is not a valid file descriptor or is not open for writing.

EINVAL
fd is attached to an object which is unsuitable for writing.

EFAULT
buf is outside your accessible address space.

EPIPE
fd is connected to a pipe or socket whose reading end is closed. When this happens the writing process will receive a **SIGPIPE** signal; if it catches, blocks or ignores this the error **EPIPE** is returned.

EAGAIN
Non-blocking I/O has been selected using **O_NONBLOCK** and there was no room in the pipe or socket connected to *fd* to write the data immediately.

EINTR
The call was interrupted by a signal before any data was written.

ENOSPC
The device containing the file referred to by *fd* has no room for the data.

EIO A low-level I/O error occurred while modifying the inode.

Other errors may occur, depending on the object connected to *fd*.

CONFORMING TO

SVr4, SVID, POSIX, X/OPEN, 4.3BSD. SVr4 documents additional error conditions EDEADLK, EFBIG, ENOLCK, ENOLNK, ENOSR, ENXIO, EPIPE, or ERANGE. Under SVr4 a write may be interrupted and return EINTR at any point, not just before any data is written.

SEE ALSO

open(2), **read**(2), **fcntl**(2), **close**(2), **lseek**(2), **select**(2), **ioctl**(2), **fsync**(2), **fwrite**(3).

TOC

Linux Man Pages Section 3

- __setfpucw.3
- abort.3
- abs.3
- acos.3
- acosh.3
- addmntent.3
- alloca.3
- alphasort.3
- asctime.3
- asin.3
- asinh.3
- assert.3
- atan.3
- atan2.3
- atanh.3
- atexit.3
- atof.3
- atoi.3
- atol.3
- bcmp.3
- bcopy.3
- bsearch.3
- bstring.3
- byteorder.3
- bzero.3
- calloc.3
- catclose.3
- catgets.3
- catopen.3
- cbrt.3
- ceil.3

- cfgetispeed.3
- cfgetospeed.3
- cfmakeraw.3
- cfsetispeed.3
- cfsetospeed.3
- clearerr.3
- clock.3
- closedir.3
- closelog.3
- confstr.3
- copysign.3
- cos.3
- cosh.3
- crypt.3
- ctermid.3
- ctime.3
- cuserid.3
- difftime.3
- div.3
- dn_comp.3
- dn_expand.3
- drand48.3
- drem.3
- ecvt.3
- endgrent.3
- endhostent.3
- endmntent.3
- endnetent.3
- endprotoent.3
- endpwent.3
- endservent.3
- endusershell.3
- endutent.3

- erand48.3
- erf.3
- erfc.3
- errno.3
- exec.3
- execl.3
- execle.3
- execlp.3
- execv.3
- execvp.3
- exit.3
- exp.3
- expm1.3
- fabs.3
- fclose.3
- fcvt.3
- fdopen.3
- feof.3
- ferror.3
- fflush.3
- ffs.3
- fgetc.3
- fgetgrent.3
- fgetpos.3
- fgetpwent.3
- fgets.3
- fileno.3
- finite.3
- floor.3
- fmod.3
- fnmatch.3
- fopen.3
- fpathconf.3

- fprintf.3
- fpurge.3
- fputc.3
- fputs.3
- fread.3
- free.3
- freopen.3
- frexp.3
- fscanf.3
- fseek.3
- fsetpos.3
- ftell.3
- ftime.3
- ftok.3
- ftw.3
- fwrite.3
- gcvt.3
- get_current_dir_name.3
- getc.3
- getchar.3
- getcwd.3
- getdirentries.3
- getenv.3
- getgrent.3
- getgrgid.3
- getgrnam.3
- gethostbyaddr.3
- gethostbyname.3
- getlogin.3
- getmntent.3
- getnetbyaddr.3
- getnetbyname.3
- getnetent.3

- getopt.3
- getopt_long.3
- getopt_long_only.3
- getpass.3
- getprotobyname.3
- getprotobynumber.3
- getprotoent.3
- getpw.3
- getpwent.3
- getpwnam.3
- getpwuid.3
- gets.3
- getservbyname.3
- getservbyport.3
- getservent.3
- getusershell.3
- getutent.3
- getutid.3
- getutline.3
- getw.3
- getwd.3
- glob.3
- globfree.3
- gmtime.3
- hasmntopt.3
- hcreate.3
- hdestroy.3
- herror.3
- hsearch.3
- htonl.3
- htons.3
- hypot.3
- index.3

- inet.3
- inet_addr.3
- inet_aton.3
- inet_lnaof.3
- inet_makeaddr.3
- inet_netof.3
- inet_network.3
- inet_ntoa.3
- infnan.3
- initgroups.3
- initstate.3
- insque.3
- intro.3
- iruserok.3
- isalnum.3
- isalpha.3
- isascii.3
- isatty.3
- isblank.3
- iscntrl.3
- isdigit.3
- isgraph.3
- isinf.3
- islower.3
- isnan.3
- isprint.3
- ispunct.3
- isspace.3
- isupper.3
- isxdigit.3
- j0.3
- j1.3
- jn.3

- jrand48.3
- killpg.3
- labs.3
- lcong48.3
- ldexp.3
- ldiv.3
- lfind.3
- lgamma.3
- localeconv.3
- localtime.3
- log.3
- log10.3
- log1p.3
- logwtmp.3
- longjmp.3
- lrand48.3
- lsearch.3
- malloc.3
- mblen.3
- mbstowcs.3
- mbtowc.3
- memccpy.3
- memchr.3
- memcmp.3
- memcpy.3
- memfrob.3
- memmem.3
- memmove.3
- memset.3
- mkfifo.3
- mkstemp.3
- mktemp.3
- mktime.3

- modf.3
- mrand48.3
- nrand48.3
- ntohl.3
- ntohs.3
- on_exit.3
- opendir.3
- openlog.3
- pathconf.3
- pclose.3
- perror.3
- popen.3
- pow.3
- printf.3
- profil.3
- psignal.3
- putc.3
- putchar.3
- putenv.3
- putpwent.3
- puts.3
- pututline.3
- putw.3
- qsort.3
- raise.3
- rand.3
- random.3
- rcmd.3
- re_comp.3
- re_exec.3
- readdir.3
- readv.3
- realloc.3

- realpath.3
- regcomp.3
- regerror.3
- regex.3
- regexec.3
- regfree.3
- remove.3
- remque.3
- res_init.3
- res_mkquery.3
- res_query.3
- res_querydomain.3
- res_search.3
- res_send.3
- resolver.3
- rewind.3
- rewinddir.3
- rindex.3
- rint.3
- rresvport.3
- ruserok.3
- scandir.3
- scanf.3
- seed48.3
- seekdir.3
- setbuf.3
- setbuffer.3
- setenv.3
- setgrent.3
- sethostent.3
- setjmp.3
- setlinebuf.3
- setlocale.3

- setmntent.3
- setnetent.3
- setprotoent.3
- setpwent.3
- setservent.3
- setstate.3
- setusershell.3
- setutent.3
- setvbuf.3
- sigaddset.3
- sigdelset.3
- sigemptyset.3
- sigfillset.3
- siginterrupt.3
- sigismember.3
- siglongjmp.3
- sigsetjmp.3
- sigsetops.3
- sin.3
- sinh.3
- sleep.3
- snprintf.3
- sprintf.3
- sqrt.3
- srand.3
- srand48.3
- srandom.3
- sscanf.3
- stdarg.3
- stderr.3
- stdin.3
- stdio.3
- stdout.3

- stpcpy.3
- strcasecmp.3
- strcat.3
- strchr.3
- strcmp.3
- strcoll.3
- strcpy.3
- strcspn.3
- strdup.3
- strerror.3
- strfry.3
- strftime.3
- string.3
- strlen.3
- strncasecmp.3
- strncat.3
- strncmp.3
- strncpy.3
- strpbrk.3
- strptime.3
- strrchr.3
- strsep.3
- strsignal.3
- strspn.3
- strstr.3
- strtod.3
- strtok.3
- strtol.3
- strtoul.3
- strxfrm.3
- swab.3
- sysconf.3
- syslog.3

- system.3
- tan.3
- tanh.3
- tcdrain.3
- tcflow.3
- tcflush.3
- tcgetattr.3
- tcgetpgrp.3
- tcsendbreak.3
- tcsetattr.3
- tcsetpgrp.3
- tdelete.3
- telldir.3
- tempnam.3
- termios.3
- tfind.3
- tmpfile.3
- tmpnam.3
- toascii.3
- tolower.3
- toupper.3
- tsearch.3
- ttyname.3
- twalk.3
- tzset.3
- ulimit.3
- undocumented.3
- ungetc.3
- unsetenv.3
- updwtmp.3
- usleep.3
- utmpname.3
- va_arg.3

- va_end.3
- va_start.3
- vfprintf.3
- vfscanf.3
- vprintf.3
- vscanf.3
- vsnprintf.3
- vsprintf.3
- vsscanf.3
- wcstombs.3
- wctomb.3
- writev.3
- y0.3
- y1.3
- yn.3

TOC

NAME

__setfpucw - set fpu control word on i386 architecture

SYNOPSIS

#include <i386/fpu_control.h>

void __setfpucw((unsigned short) *control_word*));

DESCRIPTION

__setfpucw transfer *control_word* to the registers of the fpu (floating point unit) on i386 architecture. This can be used to control floating point precision, rounding and floating point exceptions.

EXAMPLE

__setfpucw(0x1372)

Set fpu control word on i386 architecture to
- extended precision
- rounding to nearest
- exceptions on overflow, zero divide and NaN

SEE ALSO

TOC

/usr/include/i386/fpu_control.h

TOC

NAME

abort - cause abnormal program termination

SYNOPSIS

```
#include <stdlib.h>

void abort(void);
```

DESCRIPTION

The **abort()** function causes abnormal program termination unless the signal SIGABORT is caught and the signal handler does not return. If the **abort()** function causes program termination, all open streams are closed and flushed.

If the SIGABORT function is blocked or ignored, the **abort()** function will still override it.

RETURN VALUE

The **abort()** function never returns.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

sigaction(2), **exit**(3)

NAME

abs - computes the absolute value of an integer.

SYNOPSIS

```
#include <stdlib.h>

int abs(int j);
```

DESCRIPTION

The **abs()** function computes the absolute value of the integer argument **j**.

RETURN VALUE

Returns the absolute value of the integer argument.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

NOTES

TOC

Trying to take the absolute value of the most negative integer is not defined.

SEE ALSO

ceil(3), **floor**(3), **fabs**(3),

NAME

acos - arc cosine function

SYNOPSIS

```
#include <math.h>

double acos(double x);
```

DESCRIPTION

The **acos()** function calculates the arc cosine of **x**; that is the value whose cosine is **x**. If **x** falls outside the range -1 to 1, **acos()** fails and *errno* is set.

RETURN VALUE

The **acos()** function returns the arc cosine in radians and the value is mathematically defined to be between 0 and PI (inclusive).

ERRORS

EDOM **x** is out of range.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

asin(3), **atan**(3), **atan2**(3),

NAME

acosh - inverse hyperbolic cosine function

SYNOPSIS

```
#include <math.h>

double acosh(double x);
```

DESCRIPTION

The **acosh()** function calculates the inverse hyperbolic cosine of **x**; that is the value whose hyperbolic cosine is **x**. If **x** is less than 1.0, **acosh()** returns not-a-number (NaN) and *errno* is set.

ERRORS

EDOM **x** is out of range.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

asinh(3), **atanh**(3), **cosh**(3),

NAME

alloca - memory allocator

SYNOPSIS

#include <stdlib.h>

void *alloca(size_t *size*));

DESCRIPTION

The **alloca** function allocates *size* bytes of space in the stack frame of the caller. This temporary space is automatically freed on return.

RETURN VALUES

The **alloca** function returns a pointer to the beginning of the allocated space. If the allocation failed, a **NULL** pointer is returned.

CONFORMING TO

There is evidence that the **alloca** function appeared in 32v, pwb, pwb.2, 3bsd, and 4bsd. There is a man page for it in

BSD 4.3. Linux uses the GNU version.

BUGS

The **alloca** function is machine dependent.

SEE ALSO

brk(2), **pagesize**(2), **calloc**(3),

NAME

asin - arc sine function

SYNOPSIS

```
#include <math.h>

double asin(double x);
```

DESCRIPTION

The **asin()** function calculates the arc sine of **x**; that is the value whose sine is **x**. If **x** falls outside the range -1 to 1, **asin()** fails and *errno* is set.

RETURN VALUE

The **asin()** function returns the arc sine in radians and the value is mathematically defined to be between -PI/2 and PI/2 (inclusive).

ERRORS

EDOM **x** is out of range.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acos(3), **atan**(3), **atan2**(3),

NAME

asinh - inverse hyperbolic sine function

SYNOPSIS

```
#include <math.h>

double asinh(double x);
```

DESCRIPTION

The **asinh()** function calculates the inverse hyperbolic sine of **x**; that is the value whose hyperbolic sine is **x**.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acosh(3), **atanh**(3), **cosh**(3),

TOC

NAME

assert - Abort the program if assertion is false.

SYNOPSIS

#include <assert.h>

void assert (int *expression*));

DESCRIPTION

assert() prints an error message to standard output and terminates the program by calling **abort()** if **expression** is false (i.e., compares equal to zero). This only happens when the macro **NDEBUG** is undefined.

RETURN VALUE

No value is returned.

CONFORMING TO

ISO9899 (ANSI C)

BUGS

assert() is implemented as a macro; if the expression tested has side - effects, program behaviour will be different depending on whether **NDEBUG** is defined. This may create Heisenbugs which go away when debugging is turned on.

SEE ALSO

exit(3), **abort**(3)

NAME

atan - arc tangent function

SYNOPSIS

```
#include <math.h>

double atan(double x);
```

DESCRIPTION

The **atan()** function calculates the arc tangent of **x**; that is the value whose tangent is **x**.

RETURN VALUE

The **atan()** function returns the arc tangent in radians and the value is mathematically defined to be between -PI/2 and PI/2 (inclusive).

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acos(3), **asin**(3), **atan2**(3),

NAME

atan2 - arc tangent function of two variables

SYNOPSIS

```
#include <math.h>

double atan2(double y, double x);
```

DESCRIPTION

The **atan2()** function calculates the arc tangent of the two variables **x** and **y**. It is similar to calculating the arc tangent of **y** / **x**, except that the signs of both arguments are used to determine the quadrant of the result.

RETURN VALUE

The **atan2()** function returns the result in radians, which is between -PI and PI (inclusive).

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acos(3), **asin**(3), **atan**(3),

NAME

atanh - inverse hyperbolic tangent function

SYNOPSIS

```
#include <math.h>

double atanh(double x);
```

DESCRIPTION

The **atanh()** function calculates the inverse hyperbolic tangent of **x**; that is the value whose hyperbolic tangent is **x**. If the absolute value of **x** is greater than 1.0, **acosh()** returns not-a-number (NaN) and *errno* is set.

ERRORS

EDOM **x** is out of range.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

asinh(3), **acosh**(3), **cosh**(3),

TOC

NAME

atexit - register a function to be called at normal program termination.

SYNOPSIS

```
#include <stdlib.h>

int atexit(void (*function)(void));
```

DESCRIPTION

The **atexit()** function registers the given *function* to be called at normal program termination, whether via **exit**(3) or via return from the program's **main**. Functions so registered are called in the reverse order of their registration; no arguments are passed.

RETURN VALUE

The **atexit()** function returns the value 0 if successful; otherwise the value -1 is returned; *errno* is not set.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

exit(3), **on_exit**(3)

TOC

NAME

atof - convert a string to a double.

SYNOPSIS

#include <stdlib.h>

double atof(const char ***nptr**));

DESCRIPTION

The **atof()** function converts the initial portion of the string pointed to by *nptr* to double. The behaviour is the same as

strtod(nptr, (char **)NULL);

except that **atof()** does not detect errors.

RETURN VALUE

The converted value.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

atoi(3), **atol**(3), **strtod**(3),

NAME

atoi - convert a string to an integer.

SYNOPSIS

```
#include <stdlib.h>

int atoi(const char *nptr));
```

DESCRIPTION

The **atoi()** function converts the initial portion of the string pointed to by *nptr* to int. The behaviour is the same as

```
strtol(nptr, (char **)NULL, 10);
```

except that **atoi()** does not detect errors.

RETURN VALUE

The converted value.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

atof(3), **atol**(3), **strtod**(3),

NAME

atol - convert a string to a long integer.

SYNOPSIS

#include <stdlib.h>

long atol(const char * *nptr)*);

DESCRIPTION

The **atol()** function converts the initial portion of the string pointed to by *nptr* to long. The behaviour is the same as

strtol(nptr, (char **)NULL, 10);

except that **atol()** does not detect errors.

RETURN VALUE

The converted value.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

atof(3), **atoi**(3), **strtod**(3),

NAME

bcmp - compare byte strings

SYNOPSIS

#include <string.h>

int bcmp(const void * *s1*, **const void *** *s2*, **int n**

DESCRIPTION

The **bcmp()** function compares the first **n** bytes of the strings *s1* and *s2*. If the two strings are equal, **bcmp()** returns 0, otherwise it returns a non-zero result. If **n** is zero, the two strings are assumed to be equal.

RETURN VALUE

The **bcmp()** function returns 0 if the strings are equal, otherwise a non-zero result is returned.

CONFORMING TO

4.3BSD. This function is deprecated -- use **memcmp** in new programs.

SEE ALSO

memcmp(3), **strcasecmp**(3), **strcmp**(3), **strcoll**(3), **strncmp**(3), **strncasecmp**(3)

TOC

NAME

bcopy - copy byte strings

SYNOPSIS

#include <string.h>

void bcopy (const void *_src_, void *_dest_, int n

DESCRIPTION

The **bcopy()** function copies the first **n** bytes of the source string *src* to the destination string *dest*. If **n** is zero, no bytes are copied.

RETURN VALUE

The **bcopy()** function returns no value.

CONFORMING TO

4.3BSD. This function is deprecated -- use **memcpy** in new programs.

SEE ALSO

memccpy(3), **memcpy**(3), **memmove**(3),

NAME

bsearch - binary search of a sorted array.

SYNOPSIS

```
#include <stdlib.h>

void *bsearch(const void *key, const void *base, size_t nmemb
     size_t size, int (*compar)(const void *, const void *));
```

DESCRIPTION

The **bsearch()** function searches an array of *nmemb* objects, the initial member of which is pointed to by *base*, for a member that matches the object pointed to by *key*. The size of each member of the array is specified by *size*.

The contents of the array should be in ascending sorted order according to the comparison function referenced by *compar*. The *compar* routine is expected to have two arguments which point to the *key* object and to an array member, in that order, and should return an integer less than, equal to, or greater than zero if the *key* object is found, respectively, to be less than, to match, or be greater than the array member.

RETURN VALUE

The **bsearch()** function returns a pointer to a matching member of the array, or NULL if no match is found. If there are multiple elements that match the key, the element returned is unspecified.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

qsort(3)

NAME

bcmp, bcopy, bzero, memccpy, memchr, memcmp, memcpy, memfrob, memmem, memmove, memset - byte string operations

SYNOPSIS

```
#include <string.h>

int bcmp(const void *s1, const void *s2, int n

void bcopy(const void *src, void *dest, int n

void bzero(void *s, int n);

void *memccpy(void *dest, const void *src, int c

void *memchr(const void *s, int c, size_t n

int memcmp(const void *s1, const void *s2, size_t n

void *memcpy(void *dest, const void *src, size_t n

void *memfrob(void *s, size_t n);

void *memmem(const void *needle, size_t needlelen,
     const void *haystack, size_t haystacklen));

void *memmove(void *dest, const void *src, size_t n

void *memset(void *s, int c, size_t n
```

DESCRIPTION

The byte string functions perform operations on strings that are not NULL-terminated. See the individual man pages for descriptions of each function.

SEE ALSO

bcmp(3), **bcopy**(3), **bzero**(3), **memcmp**(3), **memcpy**(3), **memfrob**(3), **memmove**(3), **memset**(3)

NAME

htonl, htons, ntohl, ntohs - convert values between host and network byte order

SYNOPSIS

```
#include <netinet/in.h>

unsigned long int htonl(unsigned long int hostlong));

unsigned short int htons(unsigned short int hostshort));

unsigned long int ntohl(unsigned long int netlong));

unsigned short int ntohs(unsigned short int netshort));
```

DESCRIPTION

The **htonl()** function converts the long integer *hostlong* from host byte order to network byte order.

The **htons()** function converts the short integer *hostshort* from host byte order to network byte order.

The **ntohl()** function converts the long integer *netlong* from network byte order to host byte order.

The **ntohs()** function converts the short integer *netshort* from network byte order to host byte order.

On the i80x86 the host byte order is Least Significant Byte first, whereas the network byte order, as used on the Internet, is Most Significant Byte first.

CONFORMING TO

BSD 4.3

SEE ALSO

gethostbyname(3), **getservent**(3)

TOC

NAME

bzero - write zeros to a byte string

SYNOPSIS

```
#include <string.h>

void bzero(void *s, int n);
```

DESCRIPTION

The **bzero()** function sets the first **n** bytes of the byte string **s** to zero.

RETURN VALUE

The **bzero()** function returns no value.

CONFORMING TO

4.3BSD. This function is deprecated -- use **memset** in new programs.

SEE ALSO

memset(3), **swab**(3)

TOC

NAME

catgets - get message from a message catalog

SYNOPSIS

#include <nl_types.h>

char *catgets(nl_catd *catalog*, **int** *set_number*, **int** *message_number*, **const char *****message*));

DESCRIPTION

catgets() reads the message *message_number*, in set *set_number*, from the message catalog identified by *catalog*, where *catalog* is a catalog descriptor returned from an earlier call to **catopen**(3). The fourth argument *message* points to a default message string which will be returned by **catgets()** if the identified message catalog is not currently available. The message-text is contained in an internal buffer area and should be copied by the application if it is to be saved or modified. The return string is always terminated with a null byte.

RETURN VALUES

On success, **catgets()** returns a pointer to an internal buffer area containing the null-terminated message string. On failure, **catgets()** returns the value *message*.

NOTES

These functions are only available in libc.so.4.4.4c and above. The Jan 1987 X/Open Portability Guide specifies a more subtle error return: *message* is returned if the message catalog specified by *catalog* is not available, while an empty string is returned when the message catalog is available but does not contain the specified message. These two possible error returns seem to be discarded in XPG4.2 in favour of always returning *message*.

CONFORMING TO

XPG4.2

SEE ALSO

catopen(3), **setlocale**(3)

NAME

catopen, catclose - open/close a message catalog

SYNOPSIS

```
#include <features.h>
#include <nl_types.h>

nl_catd catopen(name, flag)
char *name;
int flag;

void catclose(catalog)
nl_catd catalog;
```

DESCRIPTION

catopen() opens a message catalog and returns a catalog descriptor. *name* specifies the name of the message catalog to be opened. If *name* specifies and absolute path, (i.e. contains a `**/**') then *name* specifies a pathname for the message catalog. Otherwise, the environment variable **NLSPATH** is used with *name* substituted for **%N** (see **locale**(7)). If **NLSPATH** does not exist in the environment, or if a message catalog cannot be opened in any of the paths specified by **NLSPATH**, then the following paths are searched in order

/etc/locale/LC_MESSAGES
/usr/lib/locale/LC_MESSAGES
/usr/lib/locale/name/LC_MESSAGES

In all cases *LC_MESSAGES* stands for the current setting of the **LC_MESSAGES** category of locale from a previous call to

setlocale() and defaults to the "C" locale. In the last search path *name* refers to the catalog name.

The *flag* argument to catopen is used to indicate the type of loading desired. This should be either **MCLoadBySet** or **MCLoadAll**. The former value indicates that only the required set from the catalog is loaded into memory when needed, whereas the latter causes the initial call to **catopen**() to load the entire catalog into memory.

catclose() closes the message catalog identified by *catalog*. It invalidates any subsequent references to the message catalog defined by *catalog*.

RETURN VALUES

catopen() returns a message catalog descriptor of type *nl_catd* on success. On failure, it returns -1.

catclose() returns 0 on success, or -1 on failure.

NOTES

These functions are only available in libc.so.4.4.4c and above. In the case of linux, the catalog descriptor *nl_catd* is actually a mmap()'ed area of memory and not a file descriptor, thus allowing catalogs to be shared.

SEE ALSO

catgets(3), **setlocale**(3)

TOC

NAME

cbrt - cube root function

SYNOPSIS

```
#include <math.h>

double cbrt ( double x );
```

DESCRIPTION

The **cbrt()** function returns the cube root of **x**. This function cannot fail; every representable real value has a representable real cube root.

CONFORMING TO

cbrt is a GNU extension.

SEE ALSO

sqrt(3), **pow**(3)

TOC

NAME

ceil - smallest integral value not less than x

SYNOPSIS

```
#include <math.h>

double ceil (double x);
```

DESCRIPTION

The **ceil()** function rounds **x** upwards to the nearest integer, returning that value as a double.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

abs(3), **fabs**(3), **floor**(3),


TOC

NAME

clock - Determine processor time

SYNOPSIS

```
#include <time.h>

clock_t clock(void);
```

DESCRIPTION

The **clock()** function returns an approximation of processor time used by the program.

RETURN VALUE

The value returned is the CPU time used so far as a **clock_t**; to get the number of seconds used, divide by **CLOCKS_PER_SEC**.

CONFORMING TO

ANSI C

BUGS

The C standard allows for arbitrary values at the start of the program; take the difference between the value returned from a call to **clock()** at the start of the program and the end to get maximum portability.

The **times()** function call returns more information.

SEE ALSO

times(2)

NAME

closedir - close a directory

SYNOPSIS

#include <sys/types.h>

#include <dirent.h>

int closedir(DIR * *dir*);

DESCRIPTION

The **closedir()** function closes the directory stream associated with *dir*. The directory stream descriptor *dir* is not available after this call.

RETURN VALUE

The **closedir()** function returns 0 on success or -1 on failure.

ERRORS

EBADF

Invalid directory stream descriptor *dir*.

CONFORMING TO

SVID 3, POSIX, BSD 4.3

SEE ALSO

close(2), **opendir**(3), **readdir**(3), **seekdir**(3), **telldir**(3), **scandir**(3)

NAME

confstr - get configuration dependent string variables

SYNOPSIS

```
#define __USE_POSIX_2
#include <unistd.h>

size_t confstr(int name, char *buf, size_t len
```

DESCRIPTION

confstr() gets the value of configuration - dependent string variables.

The *name* argument is the system variable to be queried. The following variables are supported:

_CS_PATH
A value for the **PATH** variable which indicates where all the POSIX.2 standard utilities can be found.

If *buf* is not **NULL**, and *len* is not zero, **confstr()** copies the value of the string to *buf* truncated to *len* - **1** characters if necessary, with a null character as termination. This can be detected by comparing the return value of **confstr()** against *len*.

If *len* is zero and *buf* is **NULL**, **confstr()** just returns the value as defined below.

RETURN VALUE

If *name* does not correspond to a valid configuration variable, **confstr()** returns 0.

EXAMPLES

The following code fragment determines the path where to find the POSIX.2 system utilities:

```
char *pathbuf; size_t n;

n = confstr(_CS_PATH,NULL,(size_t)0);
if ((pathbuf = malloc(n)) == NULL) abort();
confstr(_CS_PATH, pathbuf, n);
```

ERRORS

If the value of *name* is invalid, *errno* is set to **EINVAL**.

CONFORMING TO

proposed POSIX.2

BUGS

POSIX.2 is not yet an approved standard; the information in this manpage is subject to change.

SEE ALSO

sh(1), **exec**(3), **system**(3)

TOC

NAME

copysign - copy sign of a number

SYNOPSIS

```
#include <math.h>

double copysign(double x, double y);
```

DESCRIPTION

The **copysign()** function returns a value whose absolute value matches **x**, but whose sign matches that of **y**.

CONFORMING TO

BSD 4.3


TOC

NAME

cos - cosine function

SYNOPSIS

```
#include <math.h>

double cos(double x);
```

DESCRIPTION

The **cos()** function returns the cosine of **x**, where **x** is given in radians.

RETURN VALUE

The **cos()** function returns a value between -1 and 1.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acos(3), **asin**(3), **atan**(3),

NAME

cosh - hyperbolic cosine function

SYNOPSIS

```
#include <math.h>

double cosh(double x);
```

DESCRIPTION

The **cosh()** function returns the hyperbolic cosine of **x**, which is defined mathematically as (exp(x) + exp(-x)) / 2.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acosh(3), **asinh**(3), **atanh**(3),

TOC

NAME

crypt - password and data encryption

SYNOPSIS

```
#define _XOPEN_SOURCE
#include <unistd.h>

char *crypt(const char *key, const char *salt));
```

DESCRIPTION

crypt is the password encryption function. It is based on the Data Encryption Standard algorithm with variations intended (among other things) to discourage use of hardware implementations of a key search.

key is a user's typed password.

salt is a two-character string chosen from the set [**azAZ09./**]. This string is used to perturb the algorithm in one of 4096 different ways.

By taking the lowest 7 bit of each character of the *key*, a 56-bit key is obtained. This 56-bit key is used to encrypt repeatedly a constant string (usually a string consisting of all zeros). The returned value points to the encrypted password, a series of 13 printable ASCII characters (the first two characters represent the salt itself). The return value points to static data whose content is overwritten by each call.

Warning: The key space consists of 2**56 equal 7.2e16 possi-

ble values. Exhaustive searches of this key space are possible using massively parallel computers. Software, such as **crack**(1), is available which will search the portion of this key space that is generally used by humans for passwords. Hence, password selection should, at minimum, avoid common words and names. The use of a **passwd**(1) program that checks for crackable passwords during the selection process is recommended.

The DES algorithm itself has a few quirks which make the use of the **crypt**(3) interface a very poor choice for anything other than password authentication. If you are planning on using the **crypt**(3) interface for a cryptography project, don't do it: get a good book on encryption and one of the widely available DES libraries.

CONFORMING TO

SVID, X/OPEN, BSD 4.3

SEE ALSO

login(1), **passwd**(1), **encrypt**(3),

TOC

NAME

ctermid - get controlling terminal name

SYNOPSIS

```
#include <stdio.h>

char *ctermid(char *s);
```

DESCRIPTION

ctermid() returns a string which is the pathname for the current controlling terminal for this process. If **s** is **NULL**, a static buffer is used, otherwise **s** points to a buffer used to hold the terminal pathname. The symbolic constant **L_ctermid** is the maximum number of characters in the returned pathname.

RETURN VALUE

The pointer to the pathname.

CONFORMING TO

POSIX.1

BUGS

The path returned may not uniquely identify the controlling terminal; it may, for example, be **/dev/tty**.

It is not assured that the program can open the terminal.

SEE ALSO

ttyname(3)

NAME

asctime, ctime, gmtime, localtime, mktime - transform binary date and time to ASCII

SYNOPSIS

```
#include <time.h>

char *asctime(const struct tm *timeptr));

char *ctime(const time_t *timep));

struct tm *gmtime(const time_t *timep));

struct tm *localtime(const time_t *timep));

time_t mktime(struct tm *timeptr));

extern char *tzname[2];
long int timezone;
extern int daylight;
```

DESCRIPTION

The **ctime()**, **gmtime()** and **localtime()** functions all take an argument of data type *time_t* which represents calendar time. When interpreted as an absolute time value, it represents the number of seconds elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time (UTC).

The **asctime()** and **mktime()** functions both take an argument representing broken-down time which is a binary representation separated into year, month, day, etc. Broken-down time

is stored in the structure *tm* which is defined in <*time.h*> as follows:

```
struct tm
{
        int     tm_sec;         /* seconds */
        int     tm_min;         /* minutes */
        int     tm_hour;        /* hours */
        int     tm_mday;        /* day of the month */
        int     tm_mon;         /* month */
        int     tm_year;        /* year */
        int     tm_wday;        /* day of the week */
        int     tm_yday;        /* day in the year */
        int     tm_isdst;       /* daylight saving time */
};
```

The members of the *tm* structure are:

tm_sec
The number of seconds after the minute, normally in the range 0 to 59, but can be up to 61 to allow for leap seconds.

tm_min
The number of minutes after the hour, in the range 0 to 59.

tm_hour
The number of hours past midnight, in the range 0 to 23.

tm_mday
The day of the month, in the range 1 to 31.

tm_mon
The number of months since January, in the range 0 to 11.

tm_year
The number of years since 1900.

tm_wday
The number of days since Sunday, in the range 0 to 6.

tm_yday
The number of days since January 1, in the range 0 to

365.

tm_isdst
A flag that indicates whether daylight saving time is in effect at the time described. The value is positive if daylight saving time is in effect, zero if it is not, and negative if the information is not available.

The **ctime()** function converts the calendar time *timep* into a string of the form

"Wed Jun 30 21:49:08 1993\n"

The abbreviations for the days of the week are `Sun', `Mon', `Tue', `Wed', `Thu', `Fri', and `Sat'. The abbreviations for the months are `Jan', `Feb', `Mar', `Apr', `May', `Jun', `Jul', `Aug', `Sep', `Oct', `Nov', and `Dec'. The return value points to a statically allocated string which might be overwritten by subsequent calls to any of the date and time functions. The function also sets the external variable *tzname* with information about the current time zone.

The **gmtime()** function converts the calendar time *timep* to broken-down time representation, expressed in Coordinated Universal Time (UTC).

The **localtime()** function converts the calendar time *timep* to broken-time representation, expressed relative to the user's specified time zone. The function sets the external variables *tzname* with information about the current time zone, *timezone* with the difference between Coordinated Universal Time (UTC) and local standard time in seconds, and *daylight* to a non-zero value if standard US daylight savings time rules apply.

The **asctime()** function converts the broken-down time value *timeptr* into a string with the same format as **ctime()**. The return value points to a statically allocated string which might be overwritten by subsequent calls to any of the date and time functions.

The **mktime()** function converts a broken-down time structure, expressed as local time, to calendar time representation. The function ignores the specified contents of the structure members *tm_wday* and *tm_yday* and recomputes them from the


TOC

other information in the broken-down time structure. Calling **mktime()** also sets the external variable *tzname* with information about the current time zone. If the specified broken-down time cannot be represented as calendar time, **mktime()** returns a value of (time_t)(-1) and does not alter the *tm_wday* and *tm_yday* members of the broken-down time structure.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

date(1), **gettimeofday**(2), **time**(2), **difftime**(3), **strftime**(3), **newctime**(3).

NAME

difftime - calculate time difference

SYNOPSIS

```
#include <time.h>

double difftime(time_t time1, time_t time0));
```

DESCRIPTION

The **difftime()** function returns the number of seconds elapsed between time *time1* and time *time0*. The two times are specified in calendar time, which represents the time elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time (UTC).

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

date(1), **gettimeofday**(2), **time**(2), **ctime**(3), **gmtime**(3), **localtime**(3)

TOC

TOC

NAME

div - computes the quotient and remainder of integer division

SYNOPSIS

```
#include <stdlib.h>

div_t div(int numer, int denom));
```

DESCRIPTION

The **div()** function computes the value *numer/denom* and returns the quotient and remainder in a structure named *div_t* that contains two integer members named *quot* and *rem*.

RETURN VALUE

The *div_t* structure.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

ldiv(3)

TOC

NAME

drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 - generate uniformly distributed pseudo-random numbers

SYNOPSIS

```
#include <stdlib.h>

double drand48(void);

double erand48(unsigned short int xsubi[3]);

long int lrand48(void);

long int nrand48(unsigned short int xsubi[3]);

long int mrand48(void);

long int jrand48(unsigned short int xsubi[3]);

void srand48(long int seedval));

unsigned short int *seed48(unsignedshortint seed16v [3]);

void lcong48(unsigned short int param[7]);
```

DESCRIPTION

These functions generate pseudo-random numbers using the linear congruential algorithm and 48-bit integer arithmetic.

The **drand48()** and **erand48()** functions return non-negative

double-precision floating-point values uniformly distributed between [0.0, 1.0).

The **lrand48()** and **nrand48()** functions return non-negative long integers uniformly distributed between 0 and 2^31.

The **mrand48()** and **jrand48()** functions return signed long integers uniformly distributed between -2^31 and 2^31.

The **srand48()**, **seed48()** and **lcong48()** functions are initialization functions, one of which should be called before using **drand48()**, **lrand48()** or **mrand49()**. The functions **erand48()**, **nrand48()** and **jrand48()** do not require an initialization function to be called first.

All the functions work by generating a sequence of 48-bit integers, *Xi*, according to the linear congruential formula:

Xn+1 = (aXn + c) mod

The parameter **m** = 2^48, hence 48-bit integer arithmetic is performed. Unless **lcong48()** is called, **a** and **c** are given by:

a = 0x5DEECE66D
c = 0xB

The value returned by any of the functions **drand48()**, **erand48()**, **lrand48()**, **nrand48()**, **mrand48()** or **jrand48()** is computed by first generating the next 48-bit *Xi* in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, is copied from the high-order bits of *Xi* and transformed into the returned value.

The functions **drand48()**, **lrand48()** and **mrand48()** store the last 48-bit *Xi* generated in an internal buffer. The functions **erand48()**, **nrand48()** and **jrand48()** require the calling program to provide storage for the successive *Xi* values in the array argument *xsubi*. The functions are initialized by placing the initial value of *Xi* into the array before calling the function for the first time.

The initializer function **srand48()** sets the high order 32-bits of *Xi* to the argument *seedval*. The low order 16-bits are set to the arbitrary value 0x330E.

The initializer function **seed48()** sets the value of *Xi* to the 48-bit value specified in the array argument *seed16v*. The previous value of *Xi* is copied into an internal buffer and a pointer to this buffer is returned by **seed48()**.

The initialization function **lcong48()** allows the user to specify initial values for *Xi*, **a** and **c**. Array argument elements *param*[**0-2**] specify *Xi*, *param*[**3-5**] specify **a**, and *param*[**6**] specifies **c**. After **lcong48()** has been called, a subsequent call to either **srand48()** or **seed48()** will restore the standard values of **a** and **c**.

CONFORMING TO

SVID 3

NOTES

These functions are declared obsolete by SVID 3, which states that rand(3) should be used instead.

SEE ALSO

rand(3), **random**(3)

NAME

drem - floating-point remainder function

SYNOPSIS

```
#include <math.h>

double drem(double x, double y);
```

DESCRIPTION

The **drem()** function computes the remainder of dividing **x** by **y**. The return value is **x** - **n** * **y**, where **n** is the quotient of **x** / **y**, rounded to the nearest integer. If the quotient is 1/2, it is rounded to the even number.

RETURN VALUE

The **drem()** function returns the remainder, unless **y** is zero, when the function fails and *errno* is set.

ERRORS

EDOM The denominator **y** is zero.

CONFORMING TO

BSD 4.3

SEE ALSO

fmod(3)

TOC

NAME

ecvt, fcvt - convert a floating-point number to a string.

SYNOPSIS

#include <stdlib.h>

char *ecvt(double *number*, **size_t** *ndigits*, **int** **decpt* **int** **sign*));

char *fcvt(double *number*, **size_t** *ndigits*, **int** **decpt* **int** **sign*));

DESCRIPTION

The **ecvt()** function converts *number* to a NULL terminated string of *ndigits* digits, and returns a pointer to the string. The string itself does not contain a decimal point; however, the position of the decimal point relative to the start of the string is stored in *decpt*. A negative value for *decpt* means that the decimal point is to the left of the start of the string. If the sign of *number* is negative, *sign* is set to a non-zero value, otherwise it's set to 0.

The **fcvt()** function is identical to **ecvt()**, except that *ndigits* specifies the number of digits after the decimal point.

RETURN VALUE

Both the **ecvt()** and **fcvt()** functions return a pointer to a static string containing the ASCII representation of *number*. The static string is overwritten by each call to **ecvt()** or **fcvt()**.

SEE ALSO

gcvt(3), **sprintf**(3)

NAME

erf, erfc - error function and complementary error function

SYNOPSIS

```
#include <math.h>

double erf(double x);

double erfc (double x);
```

DESCRIPTION

The **erf()** function returns the error function of **x**; defined as

erf(x) = 2/sqrt(pi)* integral from 0 to x of exp(-t*t) dt

The **erfc()** function returns the complementary error function of **x**, that is 1.0 - erf(x).

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

TOC

exp(3)

TOC

NAME

errno - number of last error

SYNOPSIS

```
#include <errno.h>

extern int errno;
```

DESCRIPTION

The integer **errno** is set by system calls (and some library functions) to indicate what went wrong. Its value is significant only when the call returned an error (usually -1), and a library function that does succeed is allowed to change **errno**.

Sometimes, when -1 is also a legal return value one has to zero **errno** before the call in order to detect possible errors.

errno is defined by the ISO C standard to be a modifiable lvalue of type **int**, and must not be explicitly declared; **errno** may be a macro. **errno** is thread-local; setting it in one thread does not affect its value in any other thread.

Valid error numbers are all non-zero; **errno** is never set to zero by any library function. All the error names specified by POSIX.1 must have distinct values.

POSIX.1 (1996 edition) lists the following symbolic error names. Of these, **EDOM** and **ERANGE** are in the ISO C standard. ISO C Amendment 1 defines the additional error number **EILSEQ**

for coding errors in multibyte or wide characters.

E2BIG
Arg list too long

EACCES
Permission denied

EAGAIN
Resource temporarily unavailable

EBADF
Bad file descriptor

EBADMSG
Bad message

EBUSY
Resource busy

ECANCELED
Operation canceled

ECHILD
No child processes

EDEADLK
Resource deadlock avoided

EDOM Domain error

EEXIST
File exists

EFAULT
Bad address

EFBIG
File too large

EINPROGRESS
Operation in progress

EINTR
Interrupted function call

EINVAL
Invalid argument

EIO Input/output error

EISDIR
Is a directory

EMFILE
Too many open files

EMLINK
Too many links

EMSGSIZE
Inappropriate message buffer length

ENAMETOOLONG
Filename too long

ENFILE
Too many open files in system

ENODEV
No such device

ENOENT
No such file or directory

ENOEXEC
Exec format error

ENOLCK
No locks available

ENOMEM
Not enough space

ENOSPC
No space left on device

ENOSYS
Function not implemented

ENOTDIR

Not a directory

ENOTEMPTY
Directory not empty

ENOTSUP
Not supported

ENOTTY
Inappropriate I/O control operation

ENXIO
No such device or address

EPERM
Operation not permitted

EPIPE
Broken pipe

ERANGE
Result too large

EROFS
Read-only file system

ESPIPE
Invalid seek

ESRCH
No such process

ETIMEDOUT
Operation timed out

EXDEV
Improper link

Many other error numbers are returned by various Unix implementations. System V returns ETXTBSY (Text file busy) if one tries to exec() a file that is currently open for writing. Linux also returns this error if one tries to have a file both memory mapped with VM_DENYWRITE and open for writing.

SEE ALSO

perror(3), **strerror**(3)

NAME

execl, execlp, execle, execv, execvp - execute a file

SYNOPSIS

```
#include <unistd.h>

extern char **environ;

int execl( const char *path, const char *arg, ...);
int execlp( const char *file, const char *arg, ...);
int execle( const char *path, const char *arg , ..., char  *
const envp[]);
int execv( const char *path, char *const argv[]);
int execvp( const char *file, char *const argv[]);
```

DESCRIPTION

The **exec** family of functions replaces the current process image with a new process image. The functions described in this manual page are front-ends for the function **execve**(2). (See the manual page for **execve** for detailed information about the replacement of the current process.)

The initial argument for these functions is the pathname of a file which is to be executed.

The *const char *arg* and subsequent ellipses in the **execl**, **execlp**, and **execle** functions can be thought of as *arg0*, *arg1*, ..., *argn*. Together they describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program. The first argument, by convention, should point to the file name asso-

ciated with the file being executed. The list of arguments *must* be terminated by a **NULL** pointer.

The **execv** and **execvp** functions provide an array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the file name associated with the file begin executed. The array of pointers *must* be terminated by a **NULL** pointer.

The **execle** function also specifies the environment of the executed process by following the **NULL** pointer that terminates the list of arguments in the parameter list or the pointer to the argv array with an additional parameter. This additional parameter is an array of pointers to null-terminated strings and *must* be terminated by a **NULL** pointer. The other functions take the environment for the new process image from the external variable *environ* in the current process.

Some of these functions have special semantics.

The functions **execlp** and **execvp** will duplicate the actions of the shell in searching for an executable file if the specified file name does not contain a slash (/) character. The search path is the path specified in the environment by the **PATH** variable. If this variable isn't specified, the default path ``:/bin:/usr/bin'' is used. In addition, certain errors are treated specially.

If permission is denied for a file (the attempted **execve** returned **EACCES**), these functions will continue searching the rest of the search path. If no other file is found, however, they will return with the global variable *errno* set to **EACCES**.

If the header of a file isn't recognized (the attempted **execve** returned **ENOEXEC**), these functions will execute the shell with the path of the file as its first argument. (If this attempt fails, no further searching is done.)

RETURN VALUES

If any of the **exec** functions returns, an error will have occurred. The return value is -1, and the global variable *errno* will be set to indicate the error.

FILES

/bin/sh

ERRORS

All of these functions may fail and set *errno* for any of the errors specified for the library function **execve**(2).

SEE ALSO

sh(1), **execve**(2), **fork**(2), **environ**(5), **ptrace**(2)

COMPATIBILITY

On some other systems the default **PATH** has the current working directory listed after */bin* and */usr/bin*, as an anti-Trojan-horse measure. As of libc 5.4.7, Linux still uses the traditional "current directory first" default **PATH**.

The behavior of **execlp** and **execvp** when errors occur while attempting to execute the file is historic practice, but has not traditionally been documented and is not specified by

the POSIX standard. BSD (and possibly other systems) do an automatic sleep and retry if ETXTBSY is encountered. Linux treats it as a hard error and returns immediately.

Traditionally, the functions **execlp** and **execvp** ignored all errors except for the ones described above and **ENOMEM** and **E2BIG**, upon which they returned. They now return if any error other than the ones described above occurs.

STANDARDS

Execl, **execv**, **execle**, **execlp** and **execvp** conform to IEEE Std1003.1-88 (``POSIX.1'').

TOC

NAME

exit - cause normal program termination

SYNOPSIS

```
#include <stdlib.h>

void exit(int status));
```

DESCRIPTION

The **exit()** function causes normal program termination and the value of *status* is returned to the parent. All functions registered with **atexit()** and **on_exit()** are called in the reverse order of their registration, and all open streams are flushed and closed.

RETURN VALUE

The **exit()** function does not return.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

_exit(2), **atexit**(3), **on_exit**(3)

NAME

exp, log, log10, pow - exponential, logarithmic and power functions

SYNOPSIS

```
#include <math.h>

double exp(double x);

double log(double x);

double log10(double x);

double pow(double x, double y);
```

DESCRIPTION

The **exp()** function returns the value of e (the base of natural logarithms) raised to the power of **x**.

The **log()** function returns the natural logarithm of **x**.

The **log10()** function returns the base-10 logarithm of **x**.

The **pow()** function returns the value of **x** raised to the power of **y**.

ERRORS

The **log()** and **log10()** functions can return the following errors:

EDOM The argument **x** is negative.

ERANGE
The argument **x** is zero. The log of zero is not defined.

The **pow()** function can return the following error:

EDOM The argument **x** is negative and **y** is not an integral value. This would result in a complex number.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

sqrt(3), **cbrt**(3)

NAME

expm1, log1p - exponential minus 1, logarithm of 1 plus argument

SYNOPSIS

```
#include <math.h>

double expm1 ( double x );

double log1p ( double x );
```

DESCRIPTION

expm1(x) returns a value equivalent to `exp (**x**) - 1'. It is computed in a way that is accurate even if the value of **x** is near zero--a case where `exp (**x**) - 1' would be inaccurate due to subtraction of two numbers that are nearly equal.

log1p(x) returns a value equivalent to `log (1 + **x**)'. It is computed in a way that is accurate even if the value of **x** is near zero.

CONFORMING TO

BSD

SEE ALSO

exp(3), **log**(3)

NAME

fabs - absolute value of floating-point number

SYNOPSIS

```
#include <math.h>

double fabs(double x);
```

DESCRIPTION

The **fabs()** function returns the absolute value of the floating-point number **x**.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

abs(3), **ceil**(3), **floor**(3),


TOC

NAME

fclose - close a stream

SYNOPSIS

```
#include <stdio.h>

int fclose( FILE *stream));
```

DESCRIPTION

The **fclose** function dissociates the named *stream* from its underlying file or set of functions. If the stream was being used for output, any buffered data is written first, using **fflush**(3).

RETURN VALUES

Upon successful completion 0 is returned. Otherwise, **EOF** is returned and the global variable *errno* is set to indicate the error. In either case any further access (including another call to **fclose**()) to the stream results in undefined behaviour.

ERRORS

EBADF
The argument *stream* is not an open stream.

The **fclose** function may also fail and set *errno* for any of the errors specified for the routines **close**(2) or **fflush**(3).

SEE ALSO

close(2), **fflush**(3), **fopen**(3),

STANDARDS

The **fclose** function conforms to ANSI C3.159-1989 (``ANSI C'').

NAME

clearerr, feof, ferror, fileno - check and reset stream status

SYNOPSIS

```
#include <stdio.h>

void clearerr( FILE *stream));
int feof( FILE *stream));
int ferror( FILE *stream));
int fileno( FILE *stream));
```

DESCRIPTION

The function **clearerr** clears the end-of-file and error indicators for the stream pointed to by *stream*.

The function **feof** tests the end-of-file indicator for the stream pointed to by *stream*, returning non-zero if it is set. The end-of-file indicator can only be cleared by the function **clearerr**.

The function **ferror** tests the error indicator for the stream pointed to by *stream*, returning non-zero if it is set. The error indicator can only be reset by the **clearerr** function.

The function **fileno** examines the argument *stream* and returns its integer descriptor.

ERRORS

These functions should not fail and do not set the external variable *errno*.

SEE ALSO

open(2), **stdio**(3)

STANDARDS

The functions **clearerr**, **feof**, and **ferror** conform to C3.159-1989 (``ANSI C'').

NAME

fflush - flush a stream

SYNOPSIS

```
#include <stdio.h>

int fflush(FILE *stream));
```

DESCRIPTION

The function **fflush** forces a write of all buffered data for the given output or update *stream* via the stream's underlying write function. The open status of the stream is unaffected.

If the *stream* argument is **NULL**, **fflush** flushes *all* open output streams.

RETURN VALUES

Upon successful completion 0 is returned. Otherwise, **EOF** is returned and the global variable *errno* is set to indicate the error.

ERRORS

EBADF
Stream is not an open stream, or is not open for writing.

The function **fflush** may also fail and set *errno* for any of the errors specified for the routine **write**(2).

SEE ALSO

write(2), **fopen**(3), **fclose**(3),

CONFORMING TO

The **fflush** function conforms to ANSI C3.159-1989 (``ANSI C'').

NAME

ffs - find first bit set in a word

SYNOPSIS

```
#include <string.h>

int ffs(int i);
```

DESCRIPTION

The **ffs()** function returns the position of the first bit set in the word **i**. The least significant bit is position 1 and the most significant position 32.

RETURN VALUE

The **ffs()** function returns the position of the first bit set, or NULL if no bits are set.

CONFORMING TO

BSD 4.3

TOC

NAME

fgetgrent - get group file entry

SYNOPSIS

```
#include <grp.h>
#include <stdio.h>
#include <sys/types.h>

struct group *fgetgrent(FILE *stream));
```

DESCRIPTION

The **fgetgrent()** function returns a pointer to a structure containing the group information from the file *stream*. The first time it is called it returns the first entry; thereafter, it returns successive entries. The file *stream* must have the same format as */etc/group*.

The *group* structure is defined in <*grp.h*> as follows:

```
struct group {
        char    *gr_name;        /* group name */
        char    *gr_passwd;      /* group password */
        gid_t   gr_gid;          /* group id */
        char    **gr_mem;        /* group members */
};
```

RETURN VALUE

The **fgetgrent()** function returns the group information structure, or NULL if there are no more entries or an error occurs.

ERRORS

ENOMEM
Insufficient memory to allocate group information structure.

CONFORMING TO

SVID 3

SEE ALSO

getgrnam(3), **getgrgid**(3), **getgrent**(3), **setgrent**(3), **endgrent**(3)

NAME

fgetpwent - get password file entry

SYNOPSIS

```
#include <pwd.h>
#include <stdio.h>
#include <sys/types.h>

struct passwd *fgetpwent(FILE *stream));
```

DESCRIPTION

The **fgetpwent()** function returns a pointer to a structure containing the broken out fields of a line in the file *stream*. The first time it is called it returns the first entry; thereafter, it returns successive entries. The file *stream* must have the same format as */etc/passwd*.

The *passwd* structure is defined in <*pwd.h*> as follows:

```
struct passwd {
        char    *pw_name;       /* user name */
        char    *pw_passwd;     /* user password */
        uid_t   pw_uid;         /* user id */
        gid_t   pw_gid;         /* group id */
        char    *pw_gecos;      /* real name */
        char    *pw_dir;        /* home directory */
        char    *pw_shell;      /* shell program */
};
```

RETURN VALUE

The **fgetpwent()** function returns the passwd structure, or NULL if there are no more entries or an error occurs.

ERRORS

ENOMEM
Insufficient memory to allocate passwd structure.

FILES

/etc/passwd
password database file

CONFORMING TO

SVID 3

SEE ALSO

getpwnam(3), **getpwuid**(3), **getpwent**(3), **endpwent**(3), **getpw**(3), **putpwent**(3), **passwd**(5).

NAME

floor - largest integral value not greater than x

SYNOPSIS

```
#include <math.h>

double floor(double x);
```

DESCRIPTION

The **floor()** function rounds **x** downwards to the nearest integer, returning that value as a double.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

abs(3), **fabs**(3), **ceil**(3),

TOC

TOC

NAME

fmod - floating-point remainder function

SYNOPSIS

```
#include <math.h>

double fmod(double x, double y);
```

DESCRIPTION

The **fmod()** function computes the remainder of dividing **x** by **y**. The return value is **x** - **n** * **y**, where **n** is the quotient of **x** / **y**, rounded towards zero to an integer.

RETURN VALUE

The **fmod()** function returns the remainder, unless **y** is zero, when the function fails and *errno* is set.

ERRORS

EDOM The denominator **y** is zero.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

drem(3)

NAME

fnmatch - match filename or pathname

SYNOPSIS

#include <fnmatch.h>

int fnmatch(const char ****pattern*, const char *****string***, int** *flags*

DESCRIPTION

The **fnmatch()** function checks whether the *string* argument matches the *pattern* argument, which is a shell wildcard pattern.

The *flags* argument modifies the behaviour; it is the bitwise OR of zero or more of the following flags:

FNM_NOESCAPE
If this flag is set, treat backslash as an ordinary character, instead of an escape character.

FNM_PATHNAME
If this flag is set, match a slash in *string* only with a slash in *pattern* and not, for example, with a [] -sequence containing a slash.

FNM_PERIOD
If this flag is set, a leading period in *string* has to be matched exactly by a period in *pattern*. A period is considered to be leading if it is the first character in *string*, or if both **FNM_PATHNAME** is set and the period immediately follows a slash.

FNM_FILE_NAME
This is a GNU synonym for **FNM_PATHNAME**.

FNM_LEADING_DIR
If this flag (a GNU extension) is set, the pattern is considered to be matched if it matches an initial segment of *string* which is followed by a slash.

FNM_CASEFOLD
If this flag (a GNU extension) is set, the pattern is matched case-insensitively.

RETURN VALUE

Zero if *string* matches *pattern*, **FNM_NOMATCH** if there is no match or another non-zero value if there is an error.

CONFORMING TO

ISO/IEC 9945-2: 1993 (POSIX.2). The **FNM_FILE_NAME**, **FNM_LEADING_DIR**, and **FNM_CASEFOLD** flags are GNU extensions.

SEE ALSO

sh(1), **glob**(3), **glob**(7)


TOC

NAME

fopen, fdopen, freopen - stream open functions

SYNOPSIS

```
#include <stdio.h>

FILE *fopen (const char *path, const char *mode));
FILE *fdopen (int fildes, const char *mode));
FILE *freopen (const char  *path,  const  char  *mode,  FILE
*stream
```

DESCRIPTION

The **fopen** function opens the file whose name is the string pointed to by *path* and associates a stream with it.

The argument *mode* points to a string beginning with one of the following sequences (Additional characters may follow these sequences.):

r Open text file for reading. The stream is positioned at the beginning of the file.

r+ Open for reading and writing. The stream is positioned at the beginning of the file.

w Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.

w+ Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The

stream is positioned at the beginning of the file.

a Open for writing. The file is created if it does not exist. The stream is positioned at the end of the file.

a+ Open for reading and writing. The file is created if it does not exist. The stream is positioned at the end of the file.

The *mode* string can also include the letter ``b'' either as a third character or as a character between the characters in any of the two-character strings described above. This is strictly for compatibility with ANSI C3.159-1989 (``ANSI C'') and has no effect; the ``b'' is ignored.

Any created files will have mode **S_IRUSR|S_IWUSR|S_IRGRP|** (0666), as modified by the process' umask value (see **umask**(2).

Reads and writes may be intermixed on read/write streams in any order. Note that ANSI C requires that a file positioning function intervene between output and input, unless an input operation encounters end-of-file. (If this condition is not met, then a read is allowed to return the result of writes other than the most recent.) Therefore it is good practice (and indeed sometimes necessary under Linux) to put an **fseek** or **fgetpos** operation between write and read operations on such a stream. This operation may be an apparent no-op (as in *fseek*(..., *0L*, *SEEK_CUR)* called for its synchronizing side effect.

The **fdopen** function associates a stream with the existing file descriptor, *fildes*. The *mode* of the stream (one of the values "r", "r+", "w", "w+", "a", "a+") must be compatible with the mode of the file descriptor. The file position indicator of the new stream is set to that belonging to *fildes*, and the error and end-of-file indicators are cleared. Modes "w" or "w+" do not cause truncation of the file. The file descriptor is not dup'ed. The result of applying *fdopen* to a shared memory object is undefined.

The **freopen** function opens the file whose name is the string pointed to by *path* and associates the stream pointed to by *stream* with it. The original stream (if it exists) is

closed. The *mode* argument is used just as in the **fopen** function. The primary use of the **freopen** function is to change the file associated with a standard text stream

RETURN VALUES

Upon successful completion **fopen**, **fdopen** and **freopen** return a **FILE** pointer. Otherwise, **NULL** is returned and the global variable *errno* is set to indicate the error.

ERRORS

EINVAL
The *mode* provided to **fopen**, **fdopen**, or **freopen** was invalid.

The **fopen**, **fdopen** and **freopen** functions may also fail and set *errno* for any of the errors specified for the routine **malloc**(3).

The **fopen** function may also fail and set *errno* for any of the errors specified for the routine **open**(2).

The **fdopen** function may also fail and set *errno* for any of the errors specified for the routine **fcntl**(2).

The **freopen** function may also fail and set *errno* for any of the errors specified for the routines **open**(2), **fclose**(3) and **fflush**(3).

SEE ALSO

open(2), **fclose**(3)

STANDARDS

TOC

The **fopen** and **freopen** functions conform to ANSI C3.159-1989 (``ANSI C''). The **fdopen** function conforms to IEEE Std1003.1-1988 (``POSIX.1'').

TOC

TOC

NAME

fpathconf, pathconf - get configuration values for files

SYNOPSIS

```
#include <unistd.h>

long fpathconf(int filedes, int name));
long pathconf(char *path, int name));
```

DESCRIPTION

fpathconf() gets a value for the configuration option *name* for the open file descriptor *filedes*.

pathconf() gets a value for configuration option *name* for the file name *path*.

The corresponding macros defined in **<unistd.h>** are minimum values; if an application wants to take advantage of values which may change, a call to **fpathconf()** or **pathconf()** can be made, which may yield more liberal results.

Setting *name* equal to one of the following constants returns the following configuration options:

_PC_LINK_MAX
: returns the maximum number of links to the file. If *filedes* or *path* refer to a directory, then the value applies to the whole directory. The corresponding macro is **_POSIX_LINK_MAX**.

_PC_MAX_CANON

returns the maximum length of a formatted input line, where *filedes* or *path* must refer to a terminal. The corresponding macro is **_POSIX_MAX_CANON**.

_PC_MAX_INPUT
returns the maximum length of an input line, where *filedes* or *path* must refer to a terminal. The corresponding macro is **_POSIX_MAX_INPUT**.

_PC_NAME_MAX
returns the maximum length of a filename in the directory *path* or *filedes*. the process is allowed to create. The corresponding macro is **_POSIX_NAME_MAX**.

_PC_PATH_MAX
returns the maximum length of a relative pathname when *path* or *filedes* is the current working directory. The corresponding macro is **_POSIX_PATH_MAX**.

_PC_PIPE_BUF
returns the size of the pipe buffer, where *filedes* must refer to a pipe or FIFO and *path* must refer to a FIFO. The corresponding macro is **_POSIX_PIPE_BUF**.

_PC_CHOWN_RESTRICTED
returns nonzero if the **chown**(2) call may not be used on this file. If *filedes* or *path* refer to a directory, then this applies to all files in that directory. The corresponding macro is **_POSIX_CHOWN_RESTRICTED**.

_PC_NO_TRUNC
returns nonzero if accessing filenames longer than **_POSIX_NAME_MAX** generates an error. The corresponding macro is **_POSIX_NO_TRUNC**.

_PC_VDISABLE
returns nonzero if special character processing can be disabled, where *filedes* or *path* must refer to a terminal.

RETURN VALUE

The limit is returned, if one exists. If the system does not have a limit for the requested resource, -1 is returned, and *errno* is unchanged. If there is an error, -1 is returned, and *errno* is set to reflect the nature of the error.

CONFORMING TO

POSIX.1

NOTES

Files with name lengths longer than the value returned for *name* equal to **_PC_NAME_MAX** may exist in the given directory.

Some returned values may be huge; they are not suitable for allocating memory.

SEE ALSO

getconf(1), **statfs**(2), **open**(2),

TOC

NAME

fread, fwrite - binary stream input/output

SYNOPSIS

```
#include <stdio.h>

size_t fread( void *ptr, size_t size, size_t nmemb FILE *stream));

size_t fwrite( const void *ptr, size_t size, size_t nmemb FILE *stream));
```

DESCRIPTION

The function **fread** reads *nmemb* elements of data, each *size* bytes long, from the stream pointed to by *stream*, storing them at the location given by *ptr*.

The function **fwrite** writes *nmemb* elements of data, each *size* bytes long, to the stream pointed to by *stream*, obtaining them from the location given by *ptr*.

RETURN VALUES

fread and **fwrite** return the number of items successfully read or written (i.e., not the number of characters). If an error occurs, or the end-of-file is reached, the return value is a short item count (or zero).

fread does not distinguish between end-of-file and error, and callers must use **feof**(3) and **ferror**(3) to determine which occurred.

SEE ALSO

feof(3), **ferror**(3), **read**(2),

STANDARDS

The functions **fread** and **fwrite** conform to ANSI C3.159-1989 (``ANSI C'').

NAME

frexp - convert floating-point number to fractional and integral components

SYNOPSIS

```
#include <math.h>

double frexp(double x, int *exp));
```

DESCRIPTION

The **frexp()** function is used to split the number **x** into a normalized fraction and an exponent which is stored in *exp*.

RETURN VALUE

The **frexp()** function returns the normalized fraction. If the argument **x** is not zero, the normalized fraction is **x** times a power of two, and is always in the range 1/2 (inclusive) to 1 (exclusive). If **x** is zero, then the normalized fraction is zero and zero is stored in *exp*.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

ldexp(3), **modf**(3)

NAME

fgetpos, fseek, fsetpos, ftell, rewind - reposition a stream

SYNOPSIS

```
#include <stdio.h>

int fseek( FILE *stream, long offset, int whence
long ftell( FILE *stream));
void rewind( FILE *stream));
int fgetpos( FILE *stream, fpos_t *pos));
int fsetpos( FILE *stream, fpos_t *pos));
```

DESCRIPTION

The **fseek** function sets the file position indicator for the stream pointed to by *stream*. The new position, measured in bytes, is obtained by adding *offset* bytes to the position specified by *whence*. If *whence* is set to **SEEK_SET**, **SEEK_CUR**, or **SEEK_END**, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively. A successful call to the **fseek** function clears the end-of-file indicator for the stream and undoes any effects of the **ungetc**(3) function on the same stream.

The **ftell** function obtains the current value of the file position indicator for the stream pointed to by *stream*.

The **rewind** function sets the file position indicator for the stream pointed to by *stream* to the beginning of the file. It is equivalent to:

```
(void)fseek(stream, 0L, SEEK_SET)
```

except that the error indicator for the stream is also cleared (see **clearerr**(3)).

The **fgetpos** and **fsetpos** functions are alternate interfaces equivalent to **ftell** and **fseek** (with whence set to **SEEK_SET**), setting and storing the current value of the file offset into or from the object referenced by *pos*. On some non-UNIX systems an **fpos_t** object may be a complex object and these routines may be the only way to portably reposition a text stream.

RETURN VALUES

The **rewind** function returns no value. Upon successful completion, **fgetpos**, **fseek**, **fsetpos** return 0, and **ftell** returns the current offset. Otherwise, -1 is returned and the global variable errno is set to indicate the error.

ERRORS

EBADF
The *stream* specified is not a seekable stream.

EINVAL
The *whence* argument to **fseek** was not **SEEK_SET**, **SEEK_END**, or **SEEK_CUR**.

The function **fgetpos**, **fseek**, **fsetpos**, and **ftell** may also fail and set *errno* for any of the errors specified for the routines **fflush**(3), **fstat**(2), **lseek**(2), and **malloc**(3).

SEE ALSO

lseek(2)

STANDARDS

The **fgetpos**, **fsetpos**, **fseek**, **ftell**, and **rewind** functions conform to ANSI C3.159-1989 (``ANSI C'').

TOC

NAME

ftime - return date and time

SYNOPSIS

```
#include <sys/timeb.h>

int ftime(struct timeb *tp));
```

DESCRIPTION

Return current date and time in *tp*, which is declared as following:

```
struct timeb {
     time_t   time;
     unsigned short millitm;
     short    timezone;
     short    dstflag;
};
```

The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Day-light Saving time applies locally during the appropriate part of the year.

RETURN VALUE

This function always returns 0.

HISTORY

The *ftime* function appeared in 4.2BSD.

CONFORMING TO

BSD 4.2
Under BSD 4.3, this call is obsoleted by **gettimeofday**(2).

SEE ALSO

time(2)

TOC

NAME

ftok - convert a pathname and a project identifier to a System V IPC key

SYNOPSIS

```
# include <sys/types.h>
# include <sys/ipc.h>

key_t ftok ( char *pathname, char proj )
```

DESCRIPTION

The function converts the pathname of an existing accessible file and a project identifier into a **key_t** type System V IPC key.

RETURN VALUE

On success the return value will be the converted **key_t** value, otherwise **-1** with **errno** indicating the error as for the **stat**(2) system call.

BUGS

The generated **key_t** value is obtained **stat**-ing the disk file corresponding to *pathname* in order to get its i-node number and the minor device number of the filesystem on which the disk file resides, then by combining the 8 bit *proj* value along with the lower 16 bits of the i-node number, along with the 8 bits of the minor device number. The algorithm does not guarantee a unique key value. In fact

- Two different names linking to the same file produce same key values.

- Using the lower 16 bits of the i-node number, gives some chance (also usually small) to have same key values for file names referring to different i-nodes.

- Not discriminating among major device numbers, gives some chance of collision (also usually small) for systems with multiple disk controllers.

SEE ALSO

ipc(5), **msgget**(2), **semget**(2), **shmget**(2), **stat**(2).

NAME

ftw - file tree walk

SYNOPSIS

#include <ftw.h>

int ftw(const char ****directory*, int (*****funcptr** **)(const char *****file***, **struct stat *****sb***, **int** ***flag***

DESCRIPTION

ftw() walks through the directory tree starting from the indicated *directory*. For each found entry in the tree, it calls *funcptr* with the full pathname of the entry relative to *directory*, a pointer to a the second argument is a pointer to the **stat**(2) structure for the entry and an int, which value will be one of the following:

FTW_F	Item is a normal file
FTW_D	Item is a directory
FTW_NS	The stat failed on the item
FTW_DNR	Item is a directory which can't be read

Warning: Anything other than directories, like symbolic links, gets the **FTW_F** tag.

ftw() recursively calls itself for traversing found directories. To avoid using up all a program's file descriptors, the *depth* specifies the number of simultaneous open directories. When the depth is exceeded, **ftw**() will become slower because directories have to be closed and reopened.

To stop the tree walk, *funcptr* returns a non-zero value; this value will become the return value of **ftw**(). Other-

wise, **ftw**() will continue until it has traversed the entire tree, in which case it will return zero, or until it hits an error such as a **malloc**(3) failure, in which case it will return -1.

Because **ftw()** uses dynamic data structures, the only safe way to exit out of a tree walk is to return a non-zero value. To handle interrupts, for example, mark that the interrupt occurred and return a non-zero value-don't use **longjmp**(3) unless the program is going to terminate.

CONFORMING TO

AES, SVID2, SVID3, XPG2, XPG3, XPG4

SEE ALSO

stat(2)

NAME

gcvt - convert a floating-point number to a string.

SYNOPSIS

```
#include <stdlib.h>

char *gcvt(double number, size_t ndigit, char *buf
```

DESCRIPTION

The **gcvt()** function converts *number* to a minimal length NULL terminated ASCII string and stores the result in *buf*. It produces *ndigit* significant digits in either printf() F format or E format.

RETURN VALUE

The **gcvt()** function returns the address of the string pointed to by *buf*.

SEE ALSO

ecvt(3), **fcvt**(3), **sprintf**(3)

TOC

NAME

getcwd, get_current_dir_name, getwd - Get current working directory

SYNOPSIS

```
#include <unistd.h>

char *getcwd(char *buf, size_t size));
char *get_current_working_dir_name(void);
char *getwd(char *buf));
```

DESCRIPTION

The **getcwd()** function copies the absolute pathname of the current working directory to the array pointed to by *buf*, which is of length *size*.

If the current absolute path name would require a buffer longer than *size* elements, **NULL** is returned, and *errno* is set to **ERANGE**; an application should check for this error, and allocate a larger buffer if necessary.

As an extension to the POSIX.1 standard, **getcwd()** allocates the buffer dynamically using **malloc()** if *buf* is **NULL** on call. In this case, the allocated buffer has the length *size* unless *size* is less than zero, when *buf* is allocated as big as necessary. It is possible (and, indeed, advisable) to **free()** the buffers if they have been obtained this way.

get_current_dir_name, which is only prototyped if **__USE_GNU** is defined, will **malloc**(3) an array big enough to hold the current directory name. If the environment variable **PWD** is

set, and its value is correct, then that value will be returned.

getwd, which is only prototyped if **__USE_BSD** is defined, will not **malloc**(3) any memory. The *buf* argument should be a pointer to an array at least **PATH_MAX** bytes long. **getwd** does only return the first **PATH_MAX** bytes of the actual pathname.

RETURN VALUE

NULL on failure (for example, if the current directory is not readable), with *errno* set accordingly, and *buf* on success.

CONFORMING TO

POSIX.1

SEE ALSO

chdir(2), **free**(3), **malloc**(3).

TOC

NAME

getdirentries - get directory entries in a filesystem independent format

SYNOPSIS

#define __USE_BSD or **#define __USE_MISC**
#include <dirent.h>

ssize_t getdirentries(int *fd*, **char** ***buf**, **size_t** *nbytes* , **off_t** ***basep**));

DESCRIPTION

Read directory entries from the directory specified by *fd* into *buf*. At most *nbytes* are read. Reading starts at offset **basep*, and **basep* is updated with the new position after reading.

RETURN VALUE

getdirentries returns the number of bytes read or zero when at the end of the directory. If an error occurs, -1 is returned, and *errno* is set appropriately.

ERRORS

See the Linux library source code for details.

SEE ALSO

open(2), **lseek**(2)

NAME

getenv - get an environment variable

SYNOPSIS

```
#include <stdlib.h>

char *getenv(const char *name));
```

DESCRIPTION

The **getenv()** function searches the environment list for a string that matches the string pointed to by *name*. The strings are of the form *name* = *value*.

RETURN VALUE

The **getenv()** function returns a pointer to the value in the environment, or NULL if there is no match.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

putenv(3), **setenv**(3), **unsetenv**(3), **environ**(5)

NAME

getgrent, setgrent, endgrent - get group file entry

SYNOPSIS

```
#include <grp.h>
#include <sys/types.h>

struct group *getgrent(void);

void setgrent(void);

void endgrent(void);
```

DESCRIPTION

The **getgrent()** function returns a pointer to a structure containing the group information from */etc/group*. The first time it is called it returns the first entry; thereafter, it returns successive entries.

The **setgrent()** function rewinds the file pointer to the beginning of the */etc/group* file.

The **endgrent()** function closes the */etc/group* file.

The *group* structure is defined in <*grp.h*> as follows:

```
struct group {
        char    *gr_name;        /* group name */
        char    *gr_passwd;      /* group password */
        gid_t   gr_gid;          /* group id */
        char    **gr_mem;        /* group members */
```

```
};
```

RETURN VALUE

The **getgrent()** function returns the group information structure, or NULL if there are no more entries or an error occurs.

ERRORS

ENOMEM
Insufficient memory to allocate group information structure.

FILES

/etc/group
group database file

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

fgetgrent(3), **getgrnam**(3), **getgrgid**(3)

NAME

getgrnam, getgrgid - get group file entry

SYNOPSIS

```
#include <grp.h>
#include <sys/types.h>

struct group *getgrnam(const char *name));

struct group *getgrgid(gid_t gid));
```

DESCRIPTION

The **getgrnam()** function returns a pointer to a structure containing the group information from */etc/group* for the entry that matches the group name *name*.

The **getgrgid()** function returns a pointer to a structure containing the group information from */etc/group* for the entry that matches the group gid *gid*.

The *group* structure is defined in <*grp.h*> as follows:

```
struct group {
        char    *gr_name;        /* group name */
        char    *gr_passwd;      /* group password */
        gid_t   gr_gid;          /* group id */
        char    **gr_mem;        /* group members */
};
```

RETURN VALUE

The **getgrnam()** and **getgrgid()** functions return the group information structure, or NULL if the matching entry is not found or an error occurs.

ERRORS

ENOMEM
Insufficient memory to allocate group information structure.

FILES

/etc/group
Group database file

CONFORMING TO

SVID 3, POSIX, BSD 4.3

SEE ALSO

fgetgrent(3), **getgrent**(3), **setgrent**(3), **endgrent**(3)

NAME

gethostbyname, gethostbyaddr, sethostent, endhostent, herror - get network host entry

SYNOPSIS

```
#include <netdb.h>
extern int h_errno;

struct hostent *gethostbyname(const char *name));

#include <sys/socket.h>        /* for AF_INET
struct hostent *gethostbyaddr(const char *addr, int len, int type

void sethostent(int stayopen));

void endhostent(void);

void herror(const char *s);
```

DESCRIPTION

The **gethostbyname()** function returns a structure of type *hostent* for the given host *name*. Here *name* is either a host name, or an IPv4 address in standard dot notation, or an IPv6 address in colon (and possibly dot) notation. (See RFC 1884 for the description of IPv6 addresses.) If *name* is an IPv4 or IPv6 address, no lookup is performed and **gethostbyname**() simply copies *name* into the *h_name* field and its *struct in_addr* equivalent into the *h_addr_list*[**0**] field of the returned *hostent* structure. If *name* doesn't end in a dot and the environment variable **HOSTALIASES** is set, the alias file pointed to by **HOSTALIASES** will first be searched for *name*. (See **hostname**(7) for the file format.) The current domain and its parents are searched unless *name* ends

in a dot.

The **gethostbyaddr()** function returns a structure of type *hostent* for the given host address *addr* of length *len* and address type *type*. The only valid address type is currently AF_INET.

The **sethostent()** function specifies, if *stayopen* is true (1), that a connected TCP socket should be used for the name server queries and that the connection should remain open during successive queries. Otherwise, name server queries will use UDP datagrams.

The **endhostent()** function ends the use of a TCP connection for name server queries.

The **herror()** function prints the error message associated with the current value of *h_errno* on stderr.

The domain name queries carried out by **gethostbyname()** and **gethostbyaddr()** use a combination of any or all of the name server **named**(8), a broken out line from */etc/hosts*, and the Network Information Service (NIS or YP), depending upon the contents of the *order* line in */etc/host.conf*. (See **resolv+**(8)). The default action is to query **named**(8), followed by */etc/hosts*.

The *hostent* structure is defined in <*netdb.h*> as follows:

```
struct hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* host address type */
        int     h_length;       /* length of address */
        char    **h_addr_list;  /* list of addresses */
}
#define h_addr  h_addr_list[0]  /* for backward compatibility */
```

The members of the *hostent* structure are:

h_name
The official name of the host.

h_aliases
A zero-terminated array of alternative names for the host.

h_addrtype

The type of address; always AF_INET at present.

h_length
The length of the address in bytes.

h_addr_list
A zero-terminated array of network addresses for the host in network byte order.

h_addr
The first address in *h_addr_list* for backward compatibility.

RETURN VALUE

The **gethostbyname()** and **gethostbyaddr()** functions return the *hostent* structure or a NULL pointer if an error occurs. On error, the *h_errno* variable holds an error number.

ERRORS

The variable *h_errno* can have the following values:

HOST_NOT_FOUND
The specified host is unknown.

NO_ADDRESS
The requested name is valid but does not have an IP address.

NO_RECOVERY
A non-recoverable name server error occurred.

TRY_AGAIN
A temporary error occurred on an authoritative name server. Try again later.

FILES

/etc/host.conf
resolver configuration file

/etc/hosts
host database file

CONFORMING TO

BSD 4.3

SEE ALSO

resolver(3), **hosts**(5), **hostname**(7), **resolv+**(8), **named**(8)

TOC

NAME

getlogin, cuserid - get user name

SYNOPSIS

```
#include <unistd.h>

char * getlogin ( void );

#include <stdio.h>

char * cuserid ( char *string );
```

DESCRIPTION

getlogin returns a pointer to a string containing the name of the user logged in on the controlling terminal of the process, or a null pointer if this information cannot be determined. The string is statically allocated and might be overwritten on subsequent calls to this function or to **cuserid**.

cuserid returns a pointer to a string containing a user name associated with the effective user ID of the process. If *string* is not a null pointer, it should be an array that can hold at least **L_cuserid** characters; the string is returned in this array. Otherwise, a pointer to a string in a static area is returned. This string is statically allocated and might be overwritten on subsequent calls to this function or to **getlogin**.

The macro **L_cuserid** is an integer constant that indicates how long an array you might need to store a user name.


TOC

L_cuserid is declared in **stdio.h**.

These functions let your program identify positively the user who is running (**cuserid**) or the user who logged in this session (**getlogin**). (These can differ when setuid programs are involved.) The user cannot do anything to fool these functions.

For most purposes, it is more useful to use the environment variable **LOGNAME** to find out who the user is. This is more flexible precisely because the user can set **LOGNAME** arbitrarily.

ERRORS

ENOMEM
Insufficient memory to allocate passwd structure.

FILES

/etc/passwd password database file
/etc/utmp (or */var/adm/utmp*, or wherever your *utmp* file lives these days - the proper location depends on your libc version)

CONFORMING TO

POSIX.1. System V has a **cuserid** function which uses the real user ID rather than the effective user ID. The **cuserid** function was included in the 1988 version of POSIX, but removed from the 1990 version.

BUGS

Unfortunately, it is often rather easy to fool getlogin(). Sometimes it does not work at all, because some program messed up the utmp file. Often, it gives only the first 8 characters of the login name. The user currently logged in on the controlling tty of our program need not be the user who started it.

Nobody knows precisely what cuserid() does - avoid it in portable programs - avoid it altogether - use getpwuid(geteuid()) instead, if that is what you meant. DO NOT USE cuserid().

SEE ALSO

geteuid(2), **getuid**(2)

TOC

NAME

getmntent, setmntent, addmntent, endmntent, hasmntopt - get file system descriptor file entry

SYNOPSIS

```
#include <stdio.h>
#include <mntent.h>

FILE *setmntent(const char *filename, const char *type));

struct mntent *getmntent(FILE *filep));

int addmntent(FILE *filep, const struct mntent *mnt));

int endmntent(FILE *filep));

char *hasmntopt(const struct mntent *mnt, const char *opt));
```

DESCRIPTION

These routines are used to access the file system description file */etc/fstab* and the mounted file system description file */etc/mtab*.

The **setmntent()** function opens the file system description file *filep* and returns a file pointer which can be used by **getmntent()**. The argument *type* is the type of access required and can take the same values as the *mode* argument of **fopen**(3).

The **getmntent()** function reads the next line from the file system description file *filep* and returns a pointer to a structure containing the broken out fields from a line in the file. The pointer points to a static area of memory which is overwritten by subsequent calls to **getmntent()**.

The **addmntent()** function adds the mntent structure *mnt* to the end of the open file *filep*.

The **endmntent()** function closes the file system description file *filep*.

The **hasmntopt()** function scans the *mnt_opts* field (see below) of the mntent structure *mnt* for a substring that matches *opt*. See <*mntent.h*> for valid mount options.

The *mntent* structure is defined in <*mntent.h*> as follows:

```
struct mntent {
        char    *mnt_fsname;    /* name of mounted file system */
        char    *mnt_dir;       /* file system path prefix */
        char    *mnt_type;      /* mount type (see mntent.h) */
        char    *mnt_opts;      /* mount options (see mntent.h) */
        int     mnt_freq;       /* dump frequency in days */
        int     mnt_passno;     /* pass number on parallel fsck */
};
```

RETURN VALUE

The **getmntent()** function returns a pointer to the mntent structure or NULL on failure.

The **addmntent()** function returns 0 on success and 1 on failure.

The **endmntent()** function always returns 1.

The **hasmntopt()** function returns the address of the sub-string if a match is found and NULL otherwise.

FILES

/etc/fstab	file system description file
/etc/mtab	mounted file system description file

CONFORMING TO

BSD 4.3

SEE ALSO

fopen(3), **fstab**(5)

TOC

NAME

getnetent, getnetbyname, getnetbyaddr, setnetent, endnetent - get network entry

SYNOPSIS

```
#include <netdb.h>

struct netent *getnetent(void);

struct netent *getnetbyname(const char *name));

struct netent *getnetbyaddr(long net, int type));

void setnetent(int stayopen));

void endnetent(void);
```

DESCRIPTION

The **getnetent()** function reads the next line from the file */etc/networks* and returns a structure *netent* containing the broken out fields from the line. The */etc/networks* file is opened if necessary.

The **getnetbyname()** function returns a *netent* structure for the line from */etc/networks* that matches the network *name*.

The **getnetbyaddr()** function returns a *netent* structure for the line that matches the network number *net* of type *type*.

The **setnetent()** function opens and rewinds the */etc/networks* file. If *stayopen* is true (1), then the file will not be

closed between calls to **getnetbyname()** and **getnetbyaddr()**.

The **endservent()** function closes */etc/networks*.

The *netent* structure is defined in <*netdb.h*> as follows:

```
struct netent {
        char    *n_name;          /* official network name */
        char    **n_aliases;      /* alias list */
        int     n_addrtype;       /* net address type */
        unsigned long int n_net;  /* network number */
}
```

The members of the *netent* structure are:

n_name
The official name of the network.

n_aliases
A zero terminated list of alternative names for the network.

n_addrtype
The type of the network number; always AF_INET.

n_net
The network number in host byte order.

RETURN VALUE

The **getnetent()**, **getnetbyname()** and **getnetbyaddr()** functions return the *netent* structure, or a NULL pointer if an error occurs or the end of the file is reached.

FILES

/etc/networks	networks database file

CONFORMING TO

BSD 4.3

SEE ALSO

getprotoent(3), **getservent**(3), **networks**(5)
RFC 1101

NAME

getopt - Parse command line options

SYNOPSIS

```
#include <unistd.h>

int getopt(int argc, char * const argv[],
           const char *optstring));

extern char *optarg;
extern int optind, opterr, optopt

#include <getopt.h>

int getopt_long(int argc, char * const argv[],
           const char *optstring,
           const struct option *longopts, int *longindex));

int getopt_long_only(int argc, char * const argv[],
           const char *optstring,
           const struct option *longopts, int *longindex));
```

DESCRIPTION

The **getopt()** function parses the command line arguments. Its arguments *argc* and *argv* are the argument count and array as passed to the **main()** function on program invocation. An element of *argv* that starts with `-' (and is not exactly "-" or "--") is an option element. The characters of this element (aside from the initial `-') are option characters. If **getopt()** is called repeatedly, it returns successively each of the option characters from each of the option elements.

If **getopt()** finds another option character, it returns that character, updating the external variable *optind* and a

static variable *nextchar* so that the next call to **getopt()** can resume the scan with the following option character or *argv*-element.

If there are no more option characters, **getopt()** returns **EOF**. Then *optind* is the index in *argv* of the first *argv*-element that is not an option.

optstring is a string containing the legitimate option characters. If such a character is followed by a colon, the option requires an argument, so **getopt** places a pointer to the following text in the same *argv*-element, or the text of the following *argv*-element, in *optarg*. Two colons mean an option takes an optional arg; if there is text in the current *argv*-element, it is returned in *optarg*, otherwise *optarg* is set to zero. This is a GNU extension. If *optstring* contains **W** followed by a semicolon, then **-W foo** is treated as the long option **--foo**. (The **-W** option is reserved by POSIX.2 for implementation extensions.) This behaviour is a GNU extension, not available with libraries before GNU libc 2.

By default, **getopt()** permutes the contents of *argv* as it scans, so that eventually all the non-options are at the end. Two other modes are also implemented. If the first character of *optstring* is `+' or the environment variable POSIXLY_CORRECT is set, then option processing stops as soon as a non-option argument is encountered. If the first character of *optstring* is `-', then each non-option *argv*-element is handled as if it were the argument of an option with character code 1. (This is used by programs that were written to expect options and other *argv*-elements in any order and that care about the ordering of the two.) The special argument `--' forces an end of option-scanning regardless of the scanning mode.

If **getopt()** does not recognize an option character, it prints an error message to stderr, stores the character in *optopt*, and returns `?'. The calling program may prevent the error message by setting *opterr* to 0.

The **getopt_long()** function works like **getopt()** except that it also accepts long options, started out by two dashes. Long option names may be abbreviated if the abbreviation is unique or is an exact match for some defined option. A long option may take a parameter, of the form **--arg=param** or **--arg param**.

longopts is a pointer to the first element of an array of **struct option** declared in **<getopt.h>** as

```
struct option {
    const char *name;
    int has_arg;
    int *flag;
    int val;
};
```

The meanings of the different fields are:

name is the name of the long option.

has_arg
is: **no_argument** (or 0) if the option does not take an argument, **required_argument** (or 1) if the option requires an argument, or **optional_argument** (or 2) if the option takes an optional argument.

flag specifies how results are returned for a long option. If *flag* is **NULL**, then **getopt_long()** returns *val*. (For example, the calling program may set *val* to the equivalent short option character.) Otherwise, **getopt_long()** returns 0, and *flag* points to a variable which is set to *val* if the option is found, but left unchanged if the option is not found.

val is the value to return, or to load into the variable pointed to by *flag*.

The last element of the array has to be filled with zeroes.

If *longindex* is not **NULL**, it points to a variable which is set to the index of the long option relative to *longopts*.

getopt_long_only() is like **getopt_long()**, but `-' as well as `--' can indicate a long option. If an option that starts with `-' (not `--') doesn't match a long option, but does match a short option, it is parsed as a short option instead.


TOC

RETURN VALUE

The **getopt()** function returns the option character if the option was found successfully, `:' if there was a missing parameter for one of the options, `?' for an unknown option character, or **EOF** for the end of the option list.

getopt_long() and **getopt_long_only()** also return the option character when a short option is recognized. For a long option, they return *val* if *flag* is **NULL**, and 0 otherwise. Error and EOF returns are the same as for **getopt()**, plus `?' for an ambiguous match or an extraneous parameter.

ENVIRONMENT VARIABLES

POSIXLY_CORRECT
If this is set, then option processing stops as soon as a non-option argument is encountered.

_<PID>_GNU_nonoption_argv_flags_
This variable was used by **bash** 2.0 to communicate to GNU libc which arguments are the results of wildcard expansion and so should not be considered as options. This behaviour was removed in **bash** version 2.01, but the support remains in GNU libc.

EXAMPLE

The following example program, from the source code, illustrates the use of **getopt_long()** with most of its features.

```
#include <stdio.h>

int
main (argc, argv)
     int argc;
     char **argv;
{
  int c;
  int digit_optind = 0;
```


```
while (1)
  {
    int this_option_optind = optind ? optind : 1;
    int option_index = 0;
    static struct option long_options[] =
    {
      {"add", 1, 0, 0},
      {"append", 0, 0, 0},
      {"delete", 1, 0, 0},
      {"verbose", 0, 0, 0},
      {"create", 1, 0, 'c'},
      {"file", 1, 0, 0},
      {0, 0, 0, 0}
    };

    c = getopt_long (argc, argv, "abc:d:012",
                long_options, &option_index);
    if (c == -1)
   break;

    switch (c)
      {
      case 0:
        printf ("option %s", long_options[option_index].name);
        if (optarg)
          printf (" with arg %s", optarg);
        printf ("\n");
        break;

      case '0':
      case '1':
      case '2':
        if (digit_optind != 0 && digit_optind != this_option_optind)
          printf ("digits occur in two different argv-elements.\n");
        digit_optind = this_option_optind;
        printf ("option %c\n", c);
        break;

      case 'a':
        printf ("option a\n");
        break;

      case 'b':
        printf ("option b\n");
        break;

      case 'c':
```


```
            printf ("option c with value `%s'\n", optarg);
            break;

          case 'd':
            printf ("option d with value `%s'\n", optarg);
            break;

          case '?':
            break;

          default:
            printf ("?? getopt returned character code 0%o ??\n", c);
          }
      }

    if (optind < argc)
      {
        printf ("non-option ARGV-elements: ");
        while (optind < argc)
        printf ("%s ", argv[optind++]);
        printf ("\n");
      }

    exit (0);
  }
```

BUGS

This manpage is confusing.

The POSIX.2 specification of **getopt()** has a technical error described in POSIX.2 Interpretation 150. The GNU implementation (and probably all other implementations) implements the correct behaviour rather than that specified.

CONFORMING TO

getopt():
POSIX.2, provided the environment variable POSIXLY_CORRECT is set. Otherwise, the elements of *argv* aren't really const, because we permute them. We

pretend they're const in the prototype to be compatible with other systems.

NAME

getpass - get a password

SYNOPSIS

```
#include <pwd.h>

char *getpass( const char * prompt );
```

DESCRIPTION

The **getpass** function displays a prompt to the standard error output, and reads in a password from */dev/tty*. If this file is not accessible, **getpass** reads from the standard input.

The password may be up to 128 characters in length, including a trailing NUL. Any additional characters and the terminating newline character are discarded.

Getpass turns off character echoing and disables the generation of signals by tty special characters (interrupt by control-C, suspend by control-Z, etc.) while reading the password.

RETURN VALUES

Getpass returns a pointer to the null terminated password.

FILES

/dev/tty

SEE ALSO

crypt(3)

HISTORY

A **getpass** function appeared in Version 7 AT&T UNIX.

BUGS

The **getpass** function leaves its result in an internal static object and returns a pointer to that object. Subsequent calls to **getpass** will modify the same object.

The calling process should zero the password as soon as possible to avoid leaving the cleartext password visible in the process's address space.

TOC

NAME

getprotoent, getprotobyname, getprotobynumber, setprotoent, endprotoent - get protocol entry

SYNOPSIS

```
#include <netdb.h>

struct protoent *getprotoent(void);

struct protoent *getprotobyname(const char *name));

struct protoent *getprotobynumber(int proto));

void setprotoent(int stayopen));

void endprotoent(void);
```

DESCRIPTION

The **getprotoent()** function reads the next line from the file */etc/protocols* and returns a structure *protoent* containing the broken out fields from the line. The */etc/protocols* file is opened if necessary.

The **getprotobyname()** function returns a *protoent* structure for the line from */etc/protocols* that matches the protocol name *name*.

The **getprotobynumber()** function returns a *protoent* structure for the line that matches the protocol number *number*.

The **setprotoent()** function opens and rewinds the

/etc/protocols file. If *stayopen* is true (1), then the file will not be closed between calls to **getprotobyname()** or **getprotobynumber()**.

The **endprotoent()** function closes */etc/protocols*.

The *protoent* structure is defined in <*netdb.h*> as follows:

```
struct protoent {
        char    *p_name;        /* official protocol name */
        char    **p_aliases;    /* alias list */
        int     p_proto;        /* protocol number */
}
```

The members of the *protoent* structure are:

p_name
The official name of the protocol.

p_aliases
A zero terminated list of alternative names for the protocol.

p_proto
The protocol number.

RETURN VALUE

The **getprotoent()**, **getprotobyname()** and **getprotobynumber()** functions return the *protoent* structure, or a NULL pointer if an error occurs or the end of the file is reached.

FILES

/etc/protocols
protocol database file

CONFORMING TO

BSD 4.3

SEE ALSO

getservent(3), **getnetent**(3), **protocols**(5)

NAME

getpw - Re-construct password line entry

SYNOPSIS

```
#include <pwd.h>
#include <sys/types.h>

int getpw(uid_t uid, char *buf));
```

DESCRIPTION

The **getpw()** function re-constructs the password line entry for the given user uid *uid* in the buffer *buf*. The returned buffer contains a line of format

name:passwd:uid:gid:gecos:dir:shell

The *passwd* structure is defined in <*pwd.h*> as follows:

```
struct passwd {
        char    *pw_name;       /* user name */
        char    *pw_passwd;     /* user password */
        uid_t   pw_uid;         /* user id */
        gid_t   pw_gid;         /* group id */
        char    *pw_gecos;      /* real name */
        char    *pw_dir;        /* home directory */
        char    *pw_shell;      /* shell program */
};
```

RETURN VALUE

The **getpw()** function returns 0 on success, or -1 if an error occurs.

ERRORS

ENOMEM
Insufficient memory to allocate passwd structure.

FILES

/etc/passwd
password database file

SEE ALSO

fgetpwent(3), **getpwent**(3), **setpwent**(3), **getpwnam**(3), **getpwuid**(3), **putpwent**(3), **passwd**(5).

TOC

NAME

getpwent, setpwent, endpwent - get password file entry

SYNOPSIS

```
#include <pwd.h>
#include <sys/types.h>

struct passwd *getpwent(void);

void setpwent(void);

void endpwent(void);
```

DESCRIPTION

The **getpwent()** function returns a pointer to a structure containing the broken out fields of a line from */etc/passwd*. The first time it is called it returns the first entry; thereafter, it returns successive entries.

The **setpwent()** function rewinds the file pointer to the beginning of the */etc/passwd* file.

The **endpwent()** function closes the */etc/passwd* file.

The *passwd* structure is defined in <*pwd.h*> as follows:

```
struct passwd {
        char    *pw_name;       /* user name */
        char    *pw_passwd;     /* user password */
        uid_t   pw_uid;         /* user id */
        gid_t   pw_gid;         /* group id */
```


```
        char    *pw_gecos;      /* real name */
        char    *pw_dir;        /* home directory */
        char    *pw_shell;      /* shell program */
};
```

RETURN VALUE

The **getpwent()** function returns the *passwd* structure, or NULL if there are no more entries or an error occurs.

ERRORS

ENOMEM
Insufficient memory to allocate passwd structure.

FILES

/etc/passwd
password database file

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

fgetpwent(3), **getpwnam**(3), **getpwuid**(3), **putpwent**(3), **passwd**(5).

TOC

NAME

getpwnam, getpwuid - get password file entry

SYNOPSIS

```
#include <pwd.h>
#include <sys/types.h>

struct passwd *getpwnam(const char * name));

struct passwd *getpwuid(uid_t uid));
```

DESCRIPTION

The **getpwnam()** function returns a pointer to a structure containing the broken out fields of a line from */etc/passwd* for the entry that matches the user name *name*.

The **getpwuid()** function returns a pointer to a structure containing the broken out fields of a line from */etc/passwd* for the entry that matches the user uid *uid*.

The *passwd* structure is defined in <*pwd.h*> as follows:

```
struct passwd {
        char    *pw_name;       /* user name */
        char    *pw_passwd;     /* user password */
        uid_t   pw_uid;         /* user id */
        gid_t   pw_gid;         /* group id */
        char    *pw_gecos;      /* real name */
        char    *pw_dir;        /* home directory */
        char    *pw_shell;      /* shell program */
};
```

RETURN VALUE

The **getpwnam()** and **getpwuid()** functions return the passwd structure, or NULL if the matching entry is not found or an error occurs.

ERRORS

ENOMEM
Insufficient memory to allocate passwd structure.

FILES

/etc/passwd
password database file

CONFORMING TO

SVID 3, POSIX, BSD 4.3

SEE ALSO

fgetpwent(3), **getpwent**(3), **setpwent**(3), **getpw**(3), **putpwent**(3), **passwd**(5).

TOC

NAME

fgetc, fgets, getc, getchar, gets, ungetc - input of characters and strings

SYNOPSIS

```
#include <stdio.h>

int fgetc(FILE *stream));
char *fgets(char *s, int size, FILE *stream
int getc(FILE *stream));
int getchar(void);
char *gets(char *s);
int ungetc(int c, FILE *stream));
```

DESCRIPTION

fgetc() reads the next character from *stream* and returns it as an **unsigned char** cast to an **int**, or **EOF** on end of file or error.

getc() is equivalent to **fgetc()** except that it may be implemented as a macro which evaluates *stream* more than once.

getchar() is equivalent to **getc(***stdin***)**.

gets() reads a line from *stdin* into the buffer pointed to by **s** until either a terminating newline or **EOF**, which it replaces with **'\0'**. No check for buffer overrun is performed (see **BUGS** below).

fgets() reads in at most one less than *size* characters from *stream* and stores them into the buffer pointed to by **s**.

Reading stops after an **EOF** or a newline. If a newline is read, it is stored into the buffer. A **'\0'** is stored after the last character in the buffer.

ungetc() pushes **c** back to *stream*, cast to **unsigned char**, where it is available for subsequent read operations. Pushed - back characters will be returned in reverse order; only one pushback is guaranteed.

Calls to the functions described here can be mixed with each other and with calls to other input functions from the **stdio** library for the same input stream.

RETURN VALUES

fgetc(), **getc()** and **getchar()** return the character read as an **unsigned char** cast to an **int** or **EOF** on end of file or error.

gets() and **fgets()** return **s** on success, and **NULL** on error or when end of file occurs while no characters have been read.

ungetc() returns **c** on success, or **EOF** on error.

CONFORMING TO

ANSI - C, POSIX.1

BUGS

Because it is impossible to tell without knowing the data in advance how many characters **gets()** will read, and because **gets()** will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use **fgets()** instead.

It is not advisable to mix calls to input functions from the **stdio** library with low - level calls to **read()** for the file descriptor associated with the input stream; the results will be undefined and very probably not what you want.

SEE ALSO

read(2), **write**(2), **fopen**(3), **scanf**(3), **puts**(3), **fseek**(3),

NAME

getservent, getservbyname, getservbyport, setservent, endservent - get service entry

SYNOPSIS

```
#include <netdb.h>

struct servent *getservent(void);

struct servent *getservbyname(const char *name, const char *proto));

struct servent *getservbyport(int port, const char *proto));

void setservent(int stayopen));

void endservent(void);
```

DESCRIPTION

The **getservent()** function reads the next line from the file */etc/services* and returns a structure *servent* containing the broken out fields from the line. The */etc/services* file is opened if necessary.

The **getservbyname()** function returns a *servent* structure for the line from */etc/services* that matches the service *name* using protocol *proto*.

The **getservbyport()** function returns a *servent* structure for the line that matches the port *port* given in network byte order using protocol *proto*.

The **setservent()** function opens and rewinds the

/etc/services file. If *stayopen* is true (1), then the file will not be closed between calls to **getservbyname()** and **getservbyport()**.

The **endservent()** function closes */etc/services*.

The *servent* structure is defined in <*netdb.h*> as follows:

```
struct servent {
        char    *s_name;        /* official service name */
        char    **s_aliases;    /* alias list */
        int     s_port;         /* port number */
        char    *s_proto;       /* protocol to use */
}
```

The members of the *servent* structure are:

s_name
The official name of the service.

s_aliases
A zero terminated list of alternative names for the service.

s_port
The port number for the service given in network byte order.

s_proto
The name of the protocol to use with this service.

RETURN VALUE

The **getservent()**, **getservbyname()** and **getservbyport()** functions return the *servent* structure, or a NULL pointer if an error occurs or the end of the file is reached.

FILES

/etc/services
services database file

CONFORMING TO

BSD 4.3

SEE ALSO

getprotoent(3), **getnetent**(3), **services**(5)

TOC

TOC

NAME

getusershell, setusershell, endusershell - get legal user shells

SYNOPSIS

```
#include <unistd.h>

char *getusershell(void);

void setusershell(void);

void endusershell(void);
```

DESCRIPTION

The **getusershell()** function returns the next line from the file */etc/shells*, opening the file if necessary. The line should contain the pathname of a valid user shell. If */etc/shells* does not exist or is unreadable, **getusershell()** behaves as if */bin/sh* and */bin/csh* were listed in the file.

The **setusershell()** function rewinds */etc/shells*.

The **endusershell()** function closes */etc/shells*.

RETURN VALUE

The **getusershell()** function returns a NULL pointer on end-of-file.

FILES

/etc/shells

CONFORMING TO

BSD 4.3

SEE ALSO

shells(5)

TOC

NAME

getutent, getutid, getutline, pututline, setutent, endutent, utmpname - access utmp file entries

SYNOPSIS

```
#include <utmp.h>

struct utmp *getutent(void);
struct utmp *getutid(struct utmp *ut));
struct utmp *getutline(struct utmp *ut));

void pututline(struct utmp *ut));

void setutent(void);
void endutent(void);

void utmpname(const char *file));
```

DESCRIPTION

utmpname() sets the name of the utmp-format file for the other utmp functions to access. If **utmpname**() is not used to set the filename before the other functions are used, they assume **_PATH_UTMP**, as defined in *<paths.h>*.

setutent() rewinds the file pointer to the beginning of the utmp file. It is generally a Good Idea to call it before any of the other functions.

endutent() closes the utmp file. It should be called when the user code is done accessing the file with the other functions.

getutent() reads a line from the current file position in the utmp file. It returns a pointer to a structure containing the fields of the line.

getutid() searches forward from the current file position in the utmp file based upon *ut*. If *ut*->**ut_type** is **RUN_LVL**, **BOOT_TIME**, **NEW_TIME**, or **OLD_TIME**, **getutid**() will find the first entry whose **ut_type** field matches *ut*->**ut_type**. If *ut*->**ut_type** is one of **INIT_PROCESS**, **LOGIN_PROCESS**, **USER_PROCESS**, or **DEAD_PROCESS**, **getutid**() will find the first entry whose **ut_id** field matches *ut*->**ut_id**.

getutline() searches forward from the current file position in the utmp file. It scans entries whose ut_type is **USER_PROCESS** or **LOGIN_PROCESS** and returns the first one whose **ut_line** field matches *ut*->**ut_line**.

pututline() writes the utmp structure *ut* into the utmp file. It uses **getutid**() to search for the proper place in the file to insert the new entry. If it cannot find an appropriate slot for *ut*, **pututline**() will append the new entry to the end of the file.

RETURN VALUE

getutent(), **getutid**(), and **getutline**() return a pointer to a **static struct utmp**.

ERRORS

On error, **(struct utmp*)0** will be returned.

EXAMPLE

The following example adds and removes a utmp record, assuming it is run from within a pseudo terminal. For usage in a real application, you should check the return values of getpwuid() and ttyname().

```
#include <string.h>
#include <stdlib.h>
#include <pwd.h>
#include <unistd.h>
#include <utmp.h>

int main(int argc, char *argv[])
{
  struct utmp entry;

  system("echo before adding entry:;who");

  entry.ut_type=USER_PROCESS;
  entry.ut_pid=getpid();
  strcpy(entry.ut_line,ttyname(0)+strlen("/dev/"));
  /* only correct for ptys named /dev/tty[pqr][0-9a-z] */
  strcpy(entry.ut_id,ttyname(0)+strlen("/dev/tty"));
  time(&entry.ut_time);
  strcpy(entry.ut_user,getpwuid(getuid())->pw_name);
  memset(entry.ut_host,0,UT_HOSTSIZE);
  entry.ut_addr=0;
  setutent();
  pututline(&entry);

  system("echo after adding entry:;who");

  entry.ut_type=DEAD_PROCESS;
  memset(entry.ut_line,0,UT_LINESIZE);
  entry.ut_time=0;
  memset(entry.ut_user,0,UT_NAMESIZE);
  setutent();
  pututline(&entry);

  system("echo after removing entry:;who");

  endutent();
  return 0;
```

```
}
```

FILES

```
/var/run/utmp  database of currently logged-in users
/var/log/wtmp  database of past user logins
```

CONFORMING TO

```
XPG 2, SVID 2, Linux FSSTND 1.2
```

SEE ALSO

utmp(5)


TOC

NAME

getw, putw - input and output of words (ints)

SYNOPSIS

```
#include <stdio.h>

int getw(FILE *stream));
int putw(int w, FILE *stream));
```

DESCRIPTION

getw reads a word (that is, an **int**) from *stream*. It's provided for compatibility with SVID. We recommend you use **fread**(3) instead.

putw writes the word **w** (that is, an **int**) to *stream*. It is provided for compatibility with SVID, but we recommend you use **fwrite**(3) instead.

RETURN VALUES

Normally, **getw** returns the word read, and **putw** returns the word written. On error, they return **EOF**.

BUGS

The value returned on error is also a legitimate data value. **ferror**(3) can be used to distinguish between the two cases.

CONFORMING TO

SVID

SEE ALSO

fread(3), **fwrite**(3), **ferror**(3),

NAME

glob, globfree - find pathnames matching a pattern, free memory from glob()

SYNOPSIS

```
#include <glob.h>

int glob(const char *pattern, int flags,
         int errfunc(const char * epath, int eerrno
         glob_t *pglob));
void globfree(glob_t *pglob));
```

DESCRIPTION

The **glob()** function searches for all the pathnames matching *pattern* according to the rules used by the shell (see **glob**(7)). No tilde expansion or parameter substitution is done; if you want these, use **wordexp**(3).

The **globfree()** function frees the dynamically allocated storage from an earlier call to **glob()**.

The results of a **glob()** call are stored in the structure pointed to by *pglob*, which is a **glob_t** which is declared in **<glob.h>** and includes the following elements defined by POSIX.2 (more may be present as a GNU extension):

```
typedef struct
{
        int gl_pathc;       /* Count of paths matched so far  */
        char **gl_pathv;    /* List of matched pathnames.  */
        int gl_offs;        /* Slots to reserve in `gl_pathv'.  */
} glob_t;
```

Results are stored in dynamically allocated storage.

The parameter *flags* is made up of bitwise OR of zero or more the following symbolic constants, which modify the of behaviour of **glob()**:

GLOB_ERR
which means to return upon read error (because a directory does not have read permission, for example),

GLOB_MARK
which means to append a slash to each path which corresponds to a directory,

GLOB_NOSORT
which means don't sort the returned pathnames (they are by default),

GLOB_DOOFS
which means that *pglob->gl_offs* slots will be reserved at the beginning of the list of strings in *pglob->pathv*,

GLOB_NOCHECK
which means that, if no pattern matches, to return the original pattern,

GLOB_APPEND
which means to append to the results of a previous call. Do not set this flag on the first invocation of **glob()**.

GLOB_NOESCAPE
which means that meta characters cannot be quoted by backslashes.

The flags may also include some of the following, which are GNU extensions and not defined by POSIX.2:

GLOB_PERIOD
which means that a leading period can be matched by meta characters,

GLOB_ALTDIRFUNC
which means that alternative functions *pglob->gl_closedir*, *pglob->gl_readdir*, *pglob->gl_opendir*, *pglob->gl_lstat*, and *pglob->gl_stat* are used for file system access instead of the normal library functions,

GLOB_BRACE

which means that **csh**(1) style brace expresions **{a,b}** are expanded,

GLOB_NOMAGIC
which means that the pattern is returned if it contains no metacharacters,

GLOB_TILDE
which means that tilde expansion is carried out, and

GLOB_ONLYDIR
which means that only directories are matched.

If *errfunc* is not **NULL**, it will be called in case of an error with the arguments *epath*, a pointer to the path which failed, and *eerrno*, the value of *errno* as returned from one of the calls to **opendir()**, **readdir()**, or **stat()**. If *errfunc* returns non-zero, or if **GLOB_ERR** is set, **glob()** will terminate after the call to *errfunc*.

Upon successful return, *pglob->gl_pathc* contains the number of matched pathnames and *pglob->gl_pathv* a pointer to the list of matched pathnames. The first pointer after the last pathname is **NULL**.

It is possible to call **glob()** several times. In that case, the **GLOB_APPEND** flag has to be set in *flags* on the second and later invocations.

As a GNU extension, *pglob->gl_flags* is set to the flags specified, **or**ed with **GLOB_MAGCHAR** if any metacharacters were found.

RETURN VALUES

On successful completion, **glob()** returns zero. Other possible returns are:

GLOB_NOSPACE
for running out of memory,

GLOB_ABORTED
for a read error, and

GLOB_NOMATCH
for no found matches.

EXAMPLES

One example of use is the following code, which simulates typing **ls -l *.c ../*.c** in the shell.

```
glob_t globbuf;

globbuf.gl_offs = 2;
glob("*.c", GLOB_DOOFS, NULL, &globbuf);
glob("../*.c", GLOB_DOOFS | GLOB_APPEND, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp("ls", &globbuf.gl_pathv[0]);
```

CONFORMING TO

POSIX.2

BUGS

The **glob()** function may fail due to failure of underlying function calls, such as **malloc()** or **opendir()**. These will store their error code in *errno*.

The structure elements *gl_pathc* and *gl_offs* should be declared as **size_t**, according to POSIX.2, but are declared as **int**.

SEE ALSO

ls(1), **sh**(1), **stat**(2), **exec**(3), **malloc**(3), **opendir**(3), **readdir**(3), **wordexp**(3), **glob**(7)

TOC

NAME

hcreate, hdestroy, hsearch - hash table management

SYNOPSIS

```
#include <search.h>

ENTRY *hsearch(ENTRY item, ACTION action));

int hcreate(unsigned nel));

void hdestroy(void);
```

DESCRIPTION

These three functions allow the user to create a hash table which associates a key with any data.

First the table must be created with the function **hcreate()**. *nel* is an estimate of the number of entries in the table. **hcreate()** may adjust this value upward to improve the performance of the resulting hash table. The GNU implementation of **hsearch()** will also enlarge the table if it gets nearly full. **malloc**(3) is used to allocate space for the table.

The corresponding function **hdestroy()** frees the memory occupied by the hash table so that a new table can be constructed.

item is of type **ENTRY**, which is a typedef defined in <*search.h*> and includes these elements:

```
typedef struct entry
  {
    char *key;
```


TOC

```
        char *data;
    } ENTRY;
```

key points to the zero-terminated ASCII string which is the search key. *data* points to the data associated with that key. (A pointer to a type other than character should be cast to pointer-to-character.) **hsearch()** searches the hash table for an item with the same key as *item*, and if successful returns a pointer to it. *action* determines what **hsearch()** does after an unsuccessful search. A value of **ENTER** instructs it to insert the new item, while a value of **FIND** means to return **NULL**.

RETURN VALUE

hcreate() returns **NULL** if the hash table cannot be successfully installed.

hsearch() returns **NULL** if *action* is **ENTER** and there is insufficient memory to expand the hash table, or *action* is **FIND** and *item* cannot be found in the hash table.

CONFORMS TO

SVID, except that in SysV, the hash table cannot grow.

BUGS

The implementation can manage only one hash table at a time. Individual hash table entries can be added, but not deleted.

EXAMPLE

The following program inserts 24 items in to a hash table, then prints some of them.

```
    #include <stdio.h>
    #include <search.h>

    char *data[]={ "alpha", "bravo", "charley", "delta",
         "echo", "foxtrot", "golf", "hotel", "india", "juliette",
         "kilo", "lima", "mike", "november", "oscar", "papa",
         "quebec", "romeo", "sierra", "tango", "uniform",
         "victor", "whiskey", "x-ray", "yankee", "zulu"
     };

    int main()
    {
      ENTRY e, *ep;
      int i;

      /* start with small table, and let it grow */
      hcreate(3);
      for (i = 0; i < 24; i++)
        {
          e.key = data[i];
          /* data is just an integer, instead of a pointer
             to something */
          e.data = (char *)i;
          ep = hsearch(e, ENTER);
          /* there should be no failures */
          if(ep == NULL) {fprintf(stderr, "entry failed\n"); exit(1);}
        }
      for (i = 22; i < 26; i++)
        /* print two entries from the table, and show that
           two are not in the table */
        {
          e.key = data[i];
          ep = hsearch(e, FIND);
          printf("%9.9s -> %9.9s:%d\n", e.key, ep?ep->key:"NULL",
                 ep?(int)(ep->data):0);
        }
      return 0;

    }
```

SEE ALSO

TOC

bsearch(3), **lsearch**(3), **tsearch**(3),

TOC

NAME

hypot - Euclidean distance function

SYNOPSIS

```
#include <math.h>

double hypot(double x, double y);
```

DESCRIPTION

The **hypot()** function returns the sqrt(x*x + y*y). This is the length of the hypotenuse of a right-angle triangle with sides of length **x** and **y**, or the distance of the point (x, y) from the origin.

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

sqrt(3)

TOC

NAME

index, rindex - locate character in string

SYNOPSIS

```
#include <string.h>

char *index(const char *s, int c);

char *rindex(const char *s, int c);
```

DESCRIPTION

The **index()** function returns a pointer to the first occurrence of the character **c** in the string **s**.

The **rindex()** function returns a pointer to the last occurrence of the character **c** in the string **s**.

The terminating NULL character is considered to be a part of the strings.

RETURN VALUE

The **index()** and **rindex()** functions return a pointer to the matched character or NULL if the character is not found.

CONFORMING TO

BSD 4.3

SEE ALSO

memchr(3), **strchr**(3), **strpbrk**(3), **strsep**(3), **strspn**(3), **strstr**(3),

NAME

inet_aton, inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof - Internet address manipulation routines

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int inet_aton(const char *cp, struct in_addr *inp));

unsigned long int inet_addr(const char *cp));

unsigned long int inet_network(const char *cp));

char *inet_ntoa(struct in_addr in));

struct in_addr inet_makeaddr(int net, int host));

unsigned long int inet_lnaof(struct in_addr in));

unsigned long int inet_netof(struct in_addr in));
```

DESCRIPTION

inet_aton() converts the Internet host address *cp* from the standard numbers-and-dots notation into binary data and stores it in the structure that *inp* points to. **inet_aton** returns nonzero if the address is valid, zero if not.

The **inet_addr()** function converts the Internet host address

cp from numbers-and-dots notation into binary data in network byte order. If the input is invalid, -1 is returned. This is an *obsolete* interface to **inet_aton**, described immediately above; it is obsolete because -1 is a valid address (255.255.255.255), and **inet_aton** provides a cleaner way to indicate error return.

The **inet_network()** function extracts the network number in host byte order from the address *cp* in numbers-and-dots notation. If the input is invalid, -1 is returned.

The **inet_ntoa()** function converts the Internet host address *in* given in network byte order to a string in standard numbers-and-dots notation. The string is returned in a statically allocated buffer, which subsequent calls will overwrite.

The **inet_makeaddr()** function makes an Internet host address in network byte order by combining the network number *net* with the local address *host* in network *net*, both in local host byte order.
The **inet_lnaof()** function returns the local host address part of the Internet address *in*. The local host address is returned in local host byte order.

The **inet_netof()** function returns the network number part of the Internet Address *in*. The network number is returned in local host byte order.

The structure *in_addr* as used in **inet_ntoa()**, **inet_makeaddr()**, **inet_lnoaf()** and **inet_netof()** is defined in *netinet/in.h* as:

```
struct in_addr {
        unsigned long int s_addr;
}
```

Note that on the i80x86 the host byte order is Least Significant Byte first, whereas the network byte order, as used on the Internet, is Most Significant Byte first.

CONFORMING TO

BSD 4.3

SEE ALSO

gethostbyname(3), **getnetent**(3), **hosts**(5),

TOC

NAME

infnan - deal with infinite or not-a-number (NaN) result

SYNOPSIS

#include <math.h>

double infnan(int *error*));

DESCRIPTION

The **infnan()** function returns a suitable value for infinity and "not-a-number" (NaN) results. The value of *error* can be ERANGE to represent infinity or anything else to represent NaN. *errno* is also set.

RETURN VALUE

If *error* is ERANGE (Infinity), HUGE_VAL is returned.

If *error* is -ERANGE (-Infinity), -HUGE_VAL is returned.

If *error* is anything else, NAN is returned.

ERRORS

ERANGE
The value of *error* is positive or negative infinity.

EDOM The value of *error* is "not-a-number" (NaN).

CONFORMING TO

BSD 4.3

NAME

initgroups - initialize the supplementary group access list

SYNOPSIS

```
#include <grp.h>
#include <sys/types.h>

int initgroups(const char *user, gid_t group));
```

DESCRIPTION

The **initgroups()** function initializes the group access list by reading the group database */etc/group* and using all groups of which *user* is a member. The additional group *group* is also added to the list.

RETURN VALUE

The **initgroups()** function returns 0 on success, or -1 if an error occurs.

ERRORS

EPERM

The calling process does not have sufficient privileges.

ENOMEM
Insufficient memory to allocate group information structure.

FILES

/etc/group group database file

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

getgroups(2), **setgroups**(2)

NAME

insque, remque - insert/remove an item from a queue

SYNOPSIS

```
#include <stdlib.h>

void insque(struct qelem *elem, struct qelem *prev));
void remque(struct qelem *elem));
```

DESCRIPTION

insque() and **remque()** are functions for manipulating queues made from doubly-linked lists. Each element in this list is of type **struct qelem**

The **qelem** structure is defined as

```
struct qelem {
    struct    qelem *q_forw;
    struct    qelem *q_back;
    char      q_data[1];
};
```

insque() inserts the element pointed to by *elem* immediately after the element pointed to by *prev*, which must **NOT** be NULL.

remque() removes the element pointed to by *elem* from the doubly-linked list.

CONFORMING TO

SVR4

BUGS

The q_data field is sometimes defined to be type **char *** , and under solaris 2.x, it doesn't appear to exist at all.

The location of the prototypes for these functions differ among several versions of UNIX. Some systems place them in <search.h>, others in <string.h>. Linux places them in <stdlib.h> since that seems to make the most sense.

Some versions of UNIX (like HP-UX 10.x) do not define a **struct qelem** but rather have the arguments to **insque()** and **remque()** be of type **void *** .

NAME

intro - Introduction to library functions

DESCRIPTION

This chapter describes all library functions excluding the library functions described in chapter 2, which implement system calls. There are various function groups which can be identified by a letter which is appended to the chapter number:

(3C) These functions, the functions from chapter 2 and from chapter 3S are contained in the C standard library libc, which will be used by **cc**(1) by default.

(3S) These functions are parts of the **stdio**(3S) library. They are contained in the standard C library libc.

(3M) These functions are contained in the arithmetic library libm. They are used by the **f77**(1) FORTRAN compiler by default, but not by the **cc**(1) C compiler, which needs the option -*lm*.

(3F) These functions are part of the FORTRAN library libF77. There are no special compiler flags needed to use these functions.

(3X) Various special libraries. The manual pages documenting their functions specify the library names.

AUTHORS

Look at the header of the manual page for the author(s) and copyright conditions. Note that these can be different from page to page!

NAME

isalnum, isalpha, isascii, isblank, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit - character classification routines

SYNOPSIS

```
#include <ctype.h>

int isalnum (int c);
int isalpha (int c);
int isascii (int c);
int isblank (int c);
int iscntrl (int c);
int isdigit (int c);
int isgraph (int c);
int islower (int c);
int isprint (int c);
int ispunct (int c);
int isspace (int c);
int isupper (int c);
int isxdigit (int c);
```

DESCRIPTION

These functions check whether **c**, which must have the value of an **unsigned char** or **EOF**, falls into a certain character class according to the current locale.

isalnum()
checks for an alphanumeric character; it is equivalent to **(isalpha(c) || isdigit(c))**.

isalpha()
checks for an alphabetic character; in the standard **"C"** locale, it is equivalent to **(isupper(c) || islower(c))**. In some locales, there may be additional characters for which **isalpha()** is true--letters which are neither upper case nor lower case.

isascii()
checks whether **c** is a 7-bit *unsigned char* value that fits into the ASCII character set. This function is a BSD extension and is also an SVID extension.

isblank()
checks for a blank character; that is, a space or a tab. This function is a GNU extension.

iscntrl()
checks for a control character.

isdigit()
checks for a digit (0 through 9).

isgraph()
checks for any printable character except space.

islower()
checks for a lower - case character.

isprint()
checks for any printable character including space.

ispunct()
checks for any printable character which is not a space or an alphanumeric character.

isspace()
checks for white - space characters. In the **"" "C"** and **"" "POSIX"** locales, these are: space, form-feed (**'\f'**), newline (**'\n'**), carriage return (**'\r'**), horizontal tab (**'\t'**), and vertical tab (**'\v'**).

isupper()
checks for an uppercase letter.

isxdigit()
checks for a hexadecimal digits, i.e. one of **0 1 2 3 4**

RETURN VALUE

The values returned are nonzero if the character **c** falls into the tested class, and a zero value if not.

CONFORMING TO

ANSI - C, BSD 4.3. **isascii()** is a BSD extension and is also an SVID extension. **isblank()** is a GNU extension.

NOTE

The details of what characters belong into which class depend on the current locale. For example, **isupper()** will not recognize an A - umlaut as an uppercase letter in the default **C** locale.

SEE ALSO

tolower(3), **toupper**(3), **setlocale**(3),

TOC

NAME

isatty - does this descriptor refer to a terminal

SYNOPSIS

```
#include <unistd.h>

int isatty ( int desc );
```

DESCRIPTION

returns 1 if *desc* is an open descriptor connected to a terminal and 0 else.

CONFORMING TO

SVID, AT&T, X/OPEN, BSD 4.3

SEE ALSO

fstat(2), **ttyname**(3)

TOC

NAME

isinf, isnan, finite - test for infinity or not-a-number (NaN)

SYNOPSIS

```
#include <math.h>

int isinf(double value));

int isnan(double value));

int finite(double value));
```

DESCRIPTION

The **isinf()** function returns -1 if *value* represents negative infinity, 1 if *value* represents positive infinity, and 0 otherwise.

The **isnan()** function returns a non-zero value if *value* is "not-a-number" (NaN), and 0 otherwise.

The **finite()** function returns a non-zero value if *value* is neither infinite nor a "not-a-number" (NaN) value, and 0 otherwise.

CONFORMING TO

TOC

BSD 4.3

TOC

NAME

j0, j1, jn, y0, y1, yn - Bessel functions

SYNOPSIS

```
#include <math.h>

double j0(double x);

double j1(double x);

double jn(int n, double x);

double y0(double x);

double y1(double x);

double yn(int n, double x);
```

DESCRIPTION

The **j0()** and **j1()** functions return Bessel functions of **x** of the first kind of orders 0 and 1, respectively. The **jn()** function returns the Bessel function of **x** of the first kind of order **n**.

The **y0()** and **y1()** functions return Bessel functions of **x** of the second kind of orders 0 and 1, respectively. The **yn()** function returns the Bessel function of **x** of the second kind of order **n**.

For the functions **y0()**, **y1()** and **yn()**, the value of **x** must be positive. For negative values of **x**, these functions

return -HUGE_VAL.

CONFORMING TO

SVID 3, BSD 4.3

BUGS

There are errors of up to 2e-16 in the values returned by **j0()**, **j1()** and **jn()** for values of **x** between -8 and 8.

TOC

NAME

killpg - send signal to all members of a process group.

SYNOPSIS

#include <signal.h>

int killpg(pid_t *pidgrp*, **int** *signal)*);

DESCRIPTION

The **killpg()** function causes signal *signal* to be sent to all the processes in the process group *pidgrp* or to the processes' own process group if *pidgrp* is equal to zero.

It is equivalent to

kill(-*pidgrp*,*signal)*);

RETURN VALUE

The value returned is -1 on error, or 0 for success.

ERRORS

Errors are returned in *errno* and can be one of the following:

EINVAL
for an invalid signal,

ESRCH
for a process group which does not exist, and

EPERM
if the userid of the calling process is not equal to that of the process the signal is sent to, and the userid is not that of the superuser.

CONFORMING TO

???

SEE ALSO

kill(2), **signal**(2), **signal**(7)

NAME

labs - computes the absolute value of a long integer.

SYNOPSIS

```
#include <stdlib.h>

long int labs(long int j);
```

DESCRIPTION

The **labs()** function computes the absolute value of the long integer argument **j**.

RETURN VALUE

Returns the absolute value of the long integer argument.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

NOTES

Trying to take the absolute value of the most negative integer is not defined.

SEE ALSO

abs(3), **ceil**(3), **floor**(3),

NAME

ldexp - multiply floating-point number by integral power of 2

SYNOPSIS

```
#include <math.h>

double ldexp(double x, int exp));
```

DESCRIPTION

The **ldexp()** function returns the result of multiplying the floating-point number **x** by 2 raised to the power *exp*.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

frexp(3), **modf**(3)

TOC

TOC

NAME

ldiv - computes the quotient and remainder of long integer division.

SYNOPSIS

```
#include <stdlib.h>

ldiv_t ldiv(long int numer, long int denom));
```

DESCRIPTION

The **ldiv()** function computes the value *numer/denom* and returns the quotient and remainder in a structure named *ldiv_t* that contains two long integer members named *quot* and *rem*.

RETURN VALUE

The *ldiv_t* structure.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

div(3)

NAME

lgamma - log gamma function

SYNOPSIS

```
#include <math.h>

double lgamma(double x);
```

DESCRIPTION

The **lgamma()** function returns the log of the absolute value of the Gamma function. The sign of the Gamma function is returned in the external integer *signgam*.

For negative integer values of **x**, **lgamma()** returns HUGE_VAL and *errno* is set to ERANGE.

ERRORS

ERANGE
Invalid argument - negative integer value of **x**.

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

infnan(3)

NAME

localeconv - get numeric formatting information

SYNOPSIS

```
#include <locale.h>

struct lconv *localeconv(void);
```

DESCRIPTION

The **localeconv()** function returns a pointer to a **struct lconv** for the current locale. This structure is defined in the header-file **locale.h** and contains all values associated with the locale categories **LC_NUMERIC** and **LC_MONETARY**. Programs may also use the functions **printf()** and **strfmom()** that behave according the the actual locale in use.

CONFORMING TO

ANSI C, POSIX.1

BUGS

The **printf()** family of functions may or may not honor the

current locale.

SEE ALSO

locale(1), **localedef**(1), **strcoll**(3), **setlocale**(3), **strftime**(3), **locale**(7)

TOC

NAME

longjmp, siglongjmp - non-local jump to a saved stack context

SYNOPSIS

```
#include <setjmp.h>

void longjmp(jmp_buf env, int val));
void siglongjmp(sigjmp_buf env, int val));
```

DESCRIPTION

longjmp() and **setjmp()** are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program. **longjmp()** restores the environment saved by the last call of **setjmp()** with the corresponding *env* argument. After **longjmp()** is completed, program execution continues as if the corresponding call of **setjmp()** had just returned the value *val*. **longjmp()** cannot cause 0 to be returned. If longjmp is invoked with a second argument of 0, 1 will be returned instead.

siglongjmp() is similar to **longjmp()** except for the type of its *env* argument. If the **sigsetjmp()** call that set this *env* used a nonzero *savesigs* flag, **siglongjmp()** also restores the set of blocked signals.

RETURN VALUE

These functions never return.

CONFORMING TO

POSIX

NOTES

POSIX does not specify whether **longjmp** will restore the signal context. If you want to save and restore signal masks, use **siglongjmp**.

longjmp() and **siglongjmp()** make programs hard to understand and maintain. If possible an alternative should be used.

SEE ALSO

setjmp(3), **sigsetjmp**(3)

TOC

NAME

lfind, lsearch - linear search of an array.

SYNOPSIS

```
#include <stdlib.h>

void *lfind(const void *key, const void *base, size_t *nmemb
     size_t size, int (*compar))(const void *, const void *));

void *lsearch(const void *key, const void *base, size_t *nmemb
     size_t size, int (*compar))(const void *, const void *));
```

DESCRIPTION

lfind() and **lsearch()** perform a linear search for *key* in the array *base* which has **nmemb* elements of *size* bytes each. The comparison function referenced by *compar* is expected to have two arguments which point to the *key* object and to an array member, in that order, and which returns zero if the *key* object matches the array member, and non-zero otherwise.

If **lsearch()** does not find a matching element, then the *key* object is inserted at the end of the table, and **nmemb* is incremented.

RETURN VALUE

lfind() returns a pointer to a matching member of the array, or **NULL** if no match is found. **lsearch()** returns a pointer to a matching member of the array, or to the newly added member if no match is found.

SEE ALSO

bsearch(3), **hsearch**(3), **tsearch**(3)

TOC

NAME

calloc, malloc, free, realloc - Allocate and free dynamic memory

SYNOPSIS

```
#include <stdlib.h>

void *calloc(size_t nmemb, size_t size));
void *malloc(size_t size));
void free(void *ptr));
void *realloc(void *ptr, size_t size));
```

DESCRIPTION

calloc() allocates memory for an array of *nmemb* elements of *size* bytes each and returns a pointer to the allocated memory. The memory is set to zero.

malloc() allocates *size* bytes and returns a pointer to the allocated memory. The memory is not cleared.

free() frees the memory space pointed to by *ptr*, which must have been returned by a previous call to **malloc()**, **calloc()** or **realloc()**. Otherwise, or if **free(***ptr)*) has already been called before, undefined behaviour occurs. If *ptr* is **NULL**, no operation is performed.

realloc() changes the size of the memory block pointed to by *ptr* to *size* bytes. The contents will be unchanged to the minimum of the old and new sizes; newly allocated memory will be uninitialized. If *ptr* is **NULL**, the call is equivalent to **malloc(size)**; if size is equal to zero, the

call is equivalent to **free(***ptr)*. Unless *ptr* is **NULL**, it must have been returned by an earlier call to **malloc()**, **calloc()** or **realloc()**.

RETURN VALUES

For **calloc()** and **malloc()**, the value returned is a pointer to the allocated memory, which is suitably aligned for any kind of variable, or **NULL** if the request fails.

free() returns no value.

realloc() returns a pointer to the newly allocated memory, which is suitably aligned for any kind of variable and may be different from *ptr*, or **NULL** if the request fails or if size was equal to 0. If **realloc()** fails the original block is left untouched - it is not freed or moved.

CONFORMING TO

ANSI-C

SEE ALSO

brk(2)

NOTES

The Unix98 standard requires **malloc()**, **calloc()**, and **realloc**() to set *errno* to ENOMEM upon failure. Glibc assumes

that this is done (and the glibc versions of these routines do this); if you use a private malloc implementation that does not set *errno*, then certain library routines may fail without having a reason in *errno*.

Crashes in **malloc()**, **free()** or **realloc()** are almost always related to heap corruption, such as overflowing an allocated chunk or freeing the same pointer twice.

Recent versions of Linux libc (later than 5.4.23) and GNU libc (2.x) include a malloc implementation which is tunable via environment variables. When **MALLOC_CHECK_** is set, a special (less efficient) implementation is used which is designed to be tolerant against simple errors, such as double calls of **free()** with the same argument, or overruns of a single byte (off-by-one bugs). Not all such errors can be proteced against, however, and memory leaks can result. If **MALLOC_CHECK_** is set to 0, any detected heap corruption is silently ignored; if set to 1, a diagnostic is printed on stderr; if set to 2, **abort()** is called immediately. This can be useful because otherwise a crash may happen much later, and the true cause for the problem is then very hard to track down.

NAME

mblen - determine the number of bytes in a character

SYNOPSIS

```
#include <stdlib.h>

int mblen(const char *s, size_t n);
```

DESCRIPTION

The **mblen()** function scans the first **n** bytes of the string **s** and returns the number of bytes in a character. The **mblen()** function is equivalent to

```
mbtowc((wchat_t *)0, s, n);
```

except that the shift state of the **mbtowc()** function is not affected.

RETURN VALUE

The **mblen()** returns the number of bytes in a character or -1 if the character is invalid or 0 if **s** is a NULL string.

CONFORMING TO

SVID 3, ISO 9899

SEE ALSO

mbstowcs(3), **mbtowc**(3), **wcstombs**(3),

TOC

NAME

mbstowcs - convert a multibyte string to a wide character string.

SYNOPSIS

```
#include <stdlib.h>

size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n
```

DESCRIPTION

The **mbstowcs()** function converts a sequence of multibyte characters from the array **s** into a sequence of wide characters and stores up to **n** wide characters in the array *pwcs*.

RETURN VALUE

mbstowcs() returns the number of wide characters stored or -1 if **s** contains an invalid multibyte character.

CONFORMING TO

SVID 3, ISO 9899

SEE ALSO

mblen(3), **mbtowc**(3), **wcstombs**(3),

NAME

mbtowc - convert a multibyte character to a wide character.

SYNOPSIS

```
#include <stdlib.h>

int mbtowc(wchar_t *pwc, const char *s, size_t n
```

DESCRIPTION

The **mbtowc()** function converts a multibyte character **s**, which is no longer than **n** bytes, into a wide character and, if *pwc* is not NULL, stores the wide character in *pwc*.

RETURN VALUE

mbtowc() returns the number of bytes in the multibyte character or -1 if the multibyte character is not valid.

CONFORMING TO

SVID 3, ISO 9899

SEE ALSO

mblen(3), **mbstowcs**(3), **wcstombs**(3),

TOC

NAME

memccpy - copy memory area

SYNOPSIS

#include <string.h>

void *memccpy(void *_dest_, const void *_src_, int c

DESCRIPTION

The **memccpy()** function copies no more than **n** bytes from memory area *src* to memory area *dest*, stopping when the character **c** is found.

RETURN VALUE

The **memccpy()** function returns a pointer to the next character in *dest* after **c**, or NULL if **c** was not found in the first **n** characters of *src*.

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

bcopy(3), **memcpy**(3), **memmove**(3),

NAME

memchr - scan memory for a character

SYNOPSIS

#include <string.h>

void *memchr(const void *s, int c, size_t n

DESCRIPTION

The **memchr()** function scans the first **n** bytes of the memory area pointed to by **s** for the character **c**. The first byte to match **c** (interpreted as an unsigned character) stops the operation.

RETURN VALUE

The **memchr()** function returns a pointer to the matching byte or NULL if the character does not occur in the given memory area.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

index(3), **rindex**(3), **strchr**(3), **strrchr**(3), **strsep**(3), **strspn**(3),

NAME

memcmp - compare memory areas

SYNOPSIS

```
#include <string.h>

int memcmp(const void *s1, const void *s2, size_t n
```

DESCRIPTION

The **memcmp()** function compares the first **n** bytes of the memory areas *s1* and *s2*. It returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*.

RETURN VALUE

The **memcmp()** function returns an integer less than, equal to, or greater than zero if the first **n** bytes of *s1* is found, respectively, to be less than, to match, or be greater than the first **n** bytes of *s2*.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

bcmp(3), **strcasecmp**(3), **strcmp**(3), **strncmp**(3), **strncasecmp**(3)

NAME

memcpy - copy memory area

SYNOPSIS

```
#include <string.h>

void *memcpy(void *dest, const void *src, size_t n
```

DESCRIPTION

The **memcpy()** function copies **n** bytes from memory area *src* to memory area *dest*. The memory areas may not overlap. Use **memmove**(3) if the memory areas do overlap.

RETURN VALUE

The **memcpy()** function returns a pointer to *dest*.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

bcopy(3), **memccpy**(3), **memmove**(3),

NAME

memfrob - frobnicate (encrypt) a memory area

SYNOPSIS

```
#include <string.h>

void *memfrob(void *s, size_t n);
```

DESCRIPTION

The **memfrob()** function encrypts the first **n** bytes of the memory area **s** by exclusive-ORing each character with the number 42. The effect can be reversed by using **memfrob()** on the encrypted memory area.

Note that this function is not a proper encryption routine as the XOR constant is fixed, and is only suitable for hiding strings.

RETURN VALUE

The **memfrob()** function returns a pointer to the encrypted memory area.

CONFORMING TO

The **memfrob()** function is unique to the Linux C Library and GNU C Library.

SEE ALSO

strfry(3)

NAME

memmem - locate a substring

SYNOPSIS

```
#include <string.h>

void *memmem(const void *haystack, size_t haystacklen,
     const void *needle, size_t needlelen));
```

DESCRIPTION

The **memmem()** function finds the start of the first occurrence of the substring *needle* of length *needlelen* in the memory area *haystack* of length *haystacklen*.

RETURN VALUE

The **memmem()** function returns a pointer to the beginning of the substring, or NULL if the substring is not found.

CONFORMING TO

This function is a GNU extension.

BUGS

This function was broken in Linux libraries up to and including libc 5.0.9; there the `needle' and `haystack' arguments were interchanged, and a pointer to the end of the first occurrence of *needle* was returned. Since libc 5.0.9 is still widely used, this is a dangerous function to use. Both old and new libc's have the bug that if *needle* is empty *haystack*-1 (instead of *haystack)* is returned. And glibc (2.0.5) makes it worse, and returns a pointer to the last byte of `haystack'. Hopefully this will be fixed. For the time being, *memmem*() should not be used with an empty `needle'.

SEE ALSO

strstr(3)

NAME

memmove - copy memory area

SYNOPSIS

#include <string.h>

void *memmove(void *_dest_, const void *_src_, size_t n

DESCRIPTION

The **memmove()** function copies **n** bytes from memory area *src* to memory area *dest*. The memory areas may overlap.

RETURN VALUE

The **memmove()** function returns a pointer to *dest*.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

bcopy(3), **memccpy**(3), **memcpy**(3),

NAME

memset - fill memory with a constant byte

SYNOPSIS

```
#include <string.h>

void *memset(void *s, int c, size_t n
```

DESCRIPTION

The **memset()** function fills the first **n** bytes of the memory area pointed to by **s** with the constant byte **c**.

RETURN VALUE

The **memset()** function returns a pointer to the memory area **s**.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

bzero(3), **swab**(3)

TOC

NAME

mkfifo - make a FIFO special file (a named pipe)

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo ( const char *pathname, mode_t mode );
```

DESCRIPTION

mkfifo makes a FIFO special file with name *pathname*. *mode* specifies the FIFO's permissions. It is modified by the process's **umask** in the usual way: the permissions of the created file are **(***mode* **& ~umask)**.

A FIFO special file is similar to a pipe, except that it is created in a different way. Instead of being an anonymous communications channel, a FIFO special file is entered into the file system by calling **mkfifo**.

Once you have created a FIFO special file in this way, any process can open it for reading or writing, in the same way as an ordinary file. However, it has to be open at both ends simultaneously before you can proceed to do any input or output operations on it. Opening a FIFO for reading normally blocks until some other process opens the same FIFO for writing, and vice versa.

RETURN VALUE

The normal, successful return value from *mkfifo* is **0**. In the case of an error, **-1** is returned (in which case, *errno* is set appropriately).

ERRORS

EACCES
One of the directories in *pathname* did not allow search (execute) permission.

EEXIST
pathname already exists.

ENAMETOOLONG
Either the total length of *pathname* is greater than **PATH_MAX**, or an individual file name component has a length greater than **NAME_MAX**. In the GNU system, there is no imposed limit on overall file name length, but some file systems may place limits on the length of a component.

ENOENT
A directory component in *pathname* does not exist or is a dangling symbolic link.

ENOSPC
The directory or filesystem has no room for the new file.

ENOTDIR
A component used as a directory in *pathname* is not, in fact, a directory.

EROFS
pathname refers to a read-only filesystem.

CONFORMING TO

POSIX.1

SEE ALSO

mkfifo(1), **read**(2), **write**(2), **close**(2), **stat**(2), **umask**(2).

TOC

NAME

mkstemp - create a unique temporary file

SYNOPSIS

#include <unistd.h>

int mkstemp(char *_template_));

DESCRIPTION

The **mkstemp()** function generates a unique temporary file name from *template*. The last six characters of *template* must be XXXXXX and these are replaced with a string that makes the filename unique. The file is then created with mode read/write and permissions 0666 (glibc 2.0.6 and earlier), 0600 (glibc 2.0.7 and later).

RETURN VALUE

The **mkstemp()** function returns the file descriptor fd of the temporary file.

ERRORS

EINVAL
The last six characters of *template* were not XXXXXX.

EEXIST
The temporary file is not unique.

BUGS

The old behaviour (creating a file with mode 0666) may be a security risk, especially since other Unix flavours use 0600, and somebody might overlook this detail when porting programs.

CONFORMING TO

BSD 4.3

SEE ALSO

mktemp(3), **tmpnam**(3), **tempnam**(3),

TOC

NAME

mktemp - make a unique temporary file name

SYNOPSIS

#include <unistd.h>

char *mktemp(char * *template*));

DESCRIPTION

The **mktemp()** function generates a unique temporary file name from *template*. The last six characters of *template* must be XXXXXX and these are replaced with a string that makes the filename unique.

RETURN VALUE

The **mktemp()** function returns a pointer to *template* on success, and NULL on failure.

ERRORS

EINVAL
The last six characters of *template* were not XXXXXX.

CONFORMING TO

BSD 4.3. POSIX dictates tmpnam().

SEE ALSO

mkstemp(3), **tmpnam**(3), **tempnam**(3),

TOC

NAME

modf - extract signed integral and fractional values from floating-point number

SYNOPSIS

```
#include <math.h>

double modf(double x, double *iptr));
```

DESCRIPTION

The **modf()** function breaks the argument **x** into an integral part and a fractional part, each of which has the same sign as **x**. The integral part is stored in *iptr*.

RETURN VALUE

The **modf()** function returns the fractional part of **x**.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

frexp(3), **ldexp**(3)

TOC

NAME

on_exit - register a function to be called at normal program termination.

SYNOPSIS

```
#include <stdlib.h>

int on_exit(void (*function)(int , void *), void *arg);
```

DESCRIPTION

The **on_exit()** function registers the given *function* to be called at normal program termination, whether via **exit**(3) or via return from the program's **main**. The *function* is passed the argument to **exit**(3) and the *arg* argument from **on_exit()**.

RETURN VALUE

The **on_exit()** function returns the value 0 if successful; otherwise the value -1 is returned.

SEE ALSO

exit(3), **atexit**(3)

TOC

NAME

opendir - open a directory

SYNOPSIS

```
#include <sys/types.h>

#include <dirent.h>

DIR *opendir(const char *name));
```

DESCRIPTION

The **opendir()** function opens a directory stream corresponding to the directory *name*, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

RETURN VALUE

The **opendir()** function returns a pointer to the directory stream or NULL if an error occurred.

ERRORS

EACCES
Permission denied.

EMFILE
Too many file descriptors in use by process.

ENFILE
Too many files are currently open in the system.

ENOENT
Directory does not exist, or *name* is an empty string.

ENOMEM
Insufficient memory to complete the operation.

ENOTDIR
name is not a directory.

CONFORMING TO

SVID 3, POSIX, BSD 4.3

SEE ALSO

open(2), **readdir**(3), **closedir**(3), **seekdir**(3), **telldir**(3), **scandir**(3)

NAME

perror - print a system error message

SYNOPSIS

```
#include <stdio.h>

void perror(const char *s);

#include <errno.h>

const char *sys_errlist[];
int sys_nerr;
```

DESCRIPTION

The routine **perror()** produces a message on the standard error output, describing the last error encountered during a call to a system or library function. The argument string **s** is printed first, then a colon and a blank, then the message and a new-line. To be of most use, the argument string should include the name of the function that incurred the error. The error number is taken from the external variable *errno*, which is set when errors occur but not cleared when non-erroneous calls are made.

The global error list *sys_errlist*[] indexed by *errno* can be used to obtain the error message without the newline. The largest message number provided in the table is *sys_nerr* -1. Be careful when directly accessing this list because new error values may not have been added to *sys_errlist*[].

When a system call fails, it usually returns -1 and sets the

variable *errno* to a value describing what went wrong. (These values can be found in <*errno.h*>.) Many library functions do likewise. The function **perror()** serves to translate this error code into human-readable form. Note that *errno* is undefined after a successful library call: this call may well change this variable, even though it succeeds, for example because it internally used some other library function that failed. Thus, if a failing call is not immediately followed by a call to **perror**, the value of *errno* should be saved.

CONFORMING TO

ANSI C, BSD 4.3, POSIX, X/OPEN

SEE ALSO

strerror(3)

TOC

NAME

popen, pclose - process I/O

SYNOPSIS

```
#include <stdio.h>

FILE *popen(const char *command, const char *type));

int pclose(FILE *stream));
```

DESCRIPTION

The **popen()** function opens a process by creating a pipe, forking, and invoking the shell. Since a pipe is by definition unidirectional, the *type* argument may specify only reading or writing, not both; the resulting stream is correspondingly read-only or write-only.

The *command* argument is a pointer to a null-terminated string containing a shell command line. This command is passed to */bin/sh* using the **-c** flag; interpretation, if any, is performed by the shell. The *mode* argument is a pointer to a null-terminated string which must be either `r' for reading or `w' for writing.

The return value from **popen()** is a normal standard I/O stream in all respects save that it must be closed with **pclose()** rather than **fclose()**. Writing to such a stream writes to the standard input of the command; the command's standard output is the same as that of the process that called **popen()**, unless this is altered by the command itself. Conversely, reading from a ``popened'' stream reads

the command's standard output, and the command's standard input is the same as that of the process that called **popen**.

Note that output **popen** streams are fully buffered by default.

The **pclose** function waits for the associated process to terminate and returns the exit status of the command as returned by **wait4**.

RETURN VALUE

The **popen** function returns **NULL** if the **fork**(2) or **pipe**(2) calls fail, or if it cannot allocate memory.

The **pclose** function returns -1 if **wait4** returns an error, or some other error is detected.

ERRORS

The **popen** function does not set *errno* if memory allocation fails. If the underlying **fork()** or **pipe()** fails, *errno* is set appropriately. If the *mode* argument is invalid, and this condition is detected, *errno* is set to **EINVAL**.

If **pclose()** cannot obtain the child status, *errno* is set to **ECHILD**.

CONFORMING TO

POSIX.2

BUGS

Since the standard input of a command opened for reading shares its seek offset with the process that called **popen()**, if the original process has done a buffered read, the command's input position may not be as expected. Similarly, the output from a command opened for writing may become intermingled with that of the original process. The latter can be avoided by calling **fflush**(3) before **popen**.

Failure to execute the shell is indistinguishable from the shell's failure to execute command, or an immediate exit of the command. The only hint is an exit status of 127.

HISTORY

A **popen()** and a **pclose()** function appeared in Version 7 AT&T UNIX.

SEE ALSO

fork(2), **sh**(1), **pipe**(2), **wait4**(2), **fflush**(3), **fclose**(3), **fopen**(3), **stdio**(3), **system**(3).

TOC

TOC

NAME

printf, fprintf, sprintf, snprintf, vprintf, vfprintf, vsprintf, vsnprintf - formatted output conversion

SYNOPSIS

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format

#include <stdarg.h>

int vprintf(const char *format, va_list ap));
int vfprintf(FILE *stream, const char *format, va_list ap
int vsprintf(char *str, const char *format, va_list ap
int vsnprintf(char *str, size_t size, const char *format
```

DESCRIPTION

The **printf** family of functions produces output according to a *format* as described below. The functions **printf** and **vprintf** write output to *stdout*, the standard output stream; **fprintf** and **vfprintf** write output to the given output *stream*; **sprintf**, **snprintf**, **vsprintf** and **vsnprintf** write to the character string *str*.

These functions write the output under the control of a *format* string that specifies how subsequent arguments (or arguments accessed via the variable-length argument facilities of **stdarg**(3)) are converted for output.

These functions return the number of characters printed (not including the trailing `\0' used to end output to strings). **snprintf** and **vsnprintf** do not write more than *size* bytes (including the trailing '\0'), and return -1 if the output was truncated due to this limit.

The format string is composed of zero or more directives: ordinary characters (not %), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character %. The arguments must correspond properly (after type promotion) with the conversion specifier. After the %, the following appear in sequence:

o Zero or more of the following flags:

specifying that the value should be converted to an ``alternate form''. For **c**, **d**, **i**, **n**, **p**, **s**, and **u** conversions, this option has no effect. For **o** conversions, the precision of the number is increased to force the first character of the output string to a zero (except if a zero value is printed with an explicit precision of zero). For **x** and **X** conversions, a non-zero result has the string `0x' (or `0X' for **X** conversions) prepended to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow it (normally, a decimal point appears in the results of those conversions only if a digit follows). For **g** and **G** conversions, trailing zeros are not removed from the result as they would otherwise be.

0 specifying zero padding. For all conversions except **n**, the converted value is padded on the left with zeros rather than blanks. If a precision is given with a numeric conversion **i**, **o**, **u**, **i**, **x**, and **X**), the **0** flag is ignored.

- (a negative field width flag) indicates the converted value is to be left adjusted on the field boundary. Except for **n** conversions, the converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A - over-

rides a **0** if both are given.

' ' (a space) specifying that a blank should be left before a positive number produced by a signed conversion **e**, **E**, **f**, **g**, **G**, or **i**).

+ specifying that a sign always be placed before a number produced by a signed conversion. A **+** overrides a space if both are used.

' specifying that in a numerical argument the output is to be grouped if the locale information indicates any. Note that many versions of **gcc** cannot parse this option and will issue a warning.

o 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 flag has been given) to fill out the field width.

o An optional precision, in the form of a period (`.') followed by an optional digit string. If the digit string is omitted, the precision is taken as zero. This gives the minimum number of digits to appear for **d**, **i**, **o**, **u**, **x**, and **X** conversions, the number of digits to appear after the decimal-point for **e**, **E**, and **f** conversions, the maximum number of significant digits for **g** and **G** conversions, or the maximum number of characters to be printed from a string for **s** conversions.

o The optional character **h**, specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion corresponds to a *short int* or *unsigned short int* argument, or that a following **n** conversion corresponds to a pointer to a *short int* argument.

o The optional character **l** (ell) specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion applies to a pointer to a *long int* or *unsigned long int* argument, or that a following **n** conversion corresponds to a pointer to a *long int* argument. Linux provides a non ANSI compliant use of two **l** flags as a synonym to **q** or **L**. Thus **ll** can be used in combination with float conversions. This usage is, however, strongly discouraged.

- **o** The character **L** specifying that a following **e**, **E**, **f**, **g**, or **G** conversion corresponds to a *long double* argument, or a following **d**, **i**, **o**, **u**, **x**, or **X** conversion corresponds to a *long long* argument. Note that *long long* is not specified in *ANSI* **C** and therefore not portable to all architectures.

- **o** The optional character **q**. This is equivalent to **L**. See the STANDARDS and BUGS sections for comments on the use of **ll**, **L**, and **q**.

- **o** A **Z** character specifying that the following integer **i**, **o**, **u**, **x**, or **X**) conversion corresponds to a *size_t* argument.

- **o** A character that specifies 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* argument supplies the field width or precision. A negative field width is treated as a left adjustment flag followed by a positive field width; a negative precision is treated as though it were missing.

The conversion specifiers and their meanings are:

diouxX
The *int* (or appropriate variant) argument is converted to signed decimal and **i**), unsigned octal unsigned decimal or unsigned hexadecimal and **X**) notation. The letters **abcdef** are used for **x** conversions; the letters **ABCDEF** are used for **X** conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros.

eE The *double* argument is rounded and converted in the style where there is one digit before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero, no decimal-point character appears. An **E** conversion uses the letter **E** (rather than **e**) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is 00.

f The *double* argument is rounded and converted to decimal notation in the style where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly zero, no decimal-point character appears. If a decimal point appears, at least one digit appears before it.

g The *double* argument is converted in style **f** or **e** (or **E** for **G** conversions). The precision specifies the number of significant digits. If the precision is missing, 6 digits are given; if the precision is zero, it is treated as 1. Style **e** is used if the exponent from its conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal point appears only if it is followed by at least one digit.

c The *int* argument is converted to an *unsigned char*, and the resulting character is written.

s The argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating **NUL** character; if a precision is specified, no more than the number specified are written. If a precision is given, no null character need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating **NUL** character.

p The pointer argument is printed in hexadecimal (as if by **%#x** or **%#lx**).

n The number of characters written so far is stored into the integer indicated by the (or variant) pointer argument. No argument is converted.

% A `%' is written. No argument is converted. The complete conversion specification is `%%'.

In no case does a non-existent 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.

EXAMPLES

To print a date and time in the form `Sunday, July 3, 10:02', where *weekday* and *month* are pointers to strings:

```
#include <stdio.h>
fprintf(stdout, "%s, %s %d, %.2d:%.2d\n",
     weekday, month, day, hour, min);
```

To print to five decimal places:

```
#include <math.h>
#include <stdio.h>
fprintf(stdout, "pi = %.5f\n", 4 * atan(1.0));
```

To allocate a 128 byte string and print into it:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
char *newfmt(const char *fmt, ...)
{
          char *p;
          va_list ap;
          if ((p = malloc(128)) == NULL)
               return (NULL);
          va_start(ap, fmt);
          (void) vsnprintf(p, 128, fmt, ap);
          va_end(ap);
          return (p);
}
```

SEE ALSO

printf(1), **scanf**(3)

STANDARDS

The **fprintf**, **printf**, **sprintf**, **vprintf**, **vfprintf**, and **vsprintf** functions conform to ANSI C3.159-1989 (``ANSI C'').

The **q** flag is the *BSD 4.4* notation for *long long*, while **ll** or the usage of **L** in integer conversions is the GNU notation.

The Linux version of these functions is based on the *GNU libio* library. Take a look at the *info* documentation of *GNU libc* (*glibc-1.08)* for a more concise description.

BUGS

Some floating point conversions under Linux cause memory leaks.

All functions are fully ANSI C3.159-1989 conformant, but provide the additional flags **q**, **Z** and **'** as well as an additional behaviour of the **L** and **l** flags. The latter may be considered to be a bug, as it changes the behaviour of flags defined in ANSI C3.159-1989.

The effect of padding the **%p** format with zeros (either by the **0** flag or by specifying a precision), and the benign effect (i.e., none) of the # flag on **%n** and **%p** conversions, as well as nonsensical combinations such as are not standard; such combinations should be avoided.

Some combinations of flags defined by *ANSI* **C** are not making sense (e.g. **%Ld**). While they may have a well-defined behaviour on Linux, this need not to be so on other architectures. Therefore it usually is better not to use flags that are not defined by *ANSI* **C** at all, i.e. use **q** instead of **L** in combination with **diouxX** conversions or **ll**.

The usage of **q** is not the same as on *BSD 4.4*, as it may be used in float conversions equivalently to **L**.

Because **sprintf** and **vsprintf** assume an infinitely long string, callers must be careful not to overflow the actual space; this is often impossible to assure.

NAME

profil - execution time profile

SYNOPSIS

#include <unistd.h>

int profil(u_short ****buf*, size_t** *bufsiz***, size_t** *offset*

DESCRIPTION

This routine provides a means to find out in what areas your program spends most of its time. The argument *buf* points to *bufsiz* bytes of core. Every virtual 10 milliseconds, the user's program counter (PC) is examined: *offset* is subtracted and the result is multiplied by *scale* and divided by 65536. If the resulting value is less than *bufsiz*, then the corresponding entry in *buf* is incremented. If *buf* is NULL, profiling is disabled.

RETURN VALUE

Zero is always returned.

BUGS

profil cannot be used on a program that also uses **ITIMER_PROF** itimers.

True kernel profiling provides more accurate results. Libc 4.4 contained a kernel patch providing a system call profil.

CONFORMING TO

Similar to a call in SVr4 (but not POSIX.1).

SEE ALSO

gprof(1), **setitimer**(2), **signal**(2),

NAME

psignal - print signal message

SYNOPSIS

```
#include <signal.h>

void psignal(int sig, const char *s);

extern const char *const sys_siglist[]
```

DESCRIPTION

The **psignal()** function displays a message on *stderr* consisting of the string **s**, a colon, a space, and a string describing the signal number *sig*. If *sig* is invalid, the message displayed will indicate an unknown signal.

The array *sys_siglist* holds the signal description strings indexed by signal number.

RETURN VALUE

The **psignal()** function returns no value.

CONFORMING TO

BSD 4.3

SEE ALSO

perror(3), **strsignal**(3)

NAME

putenv - change or add an environment variable

SYNOPSIS

```
#include <stdlib.h>

int putenv(const char *string));
```

DESCRIPTION

The **putenv()** function adds or changes the value of environment variables. The argument *string* is of the form *name=value*. If *name* does not already exist in the environment, then *string* is added to the environment. If *name* does exist, then the value of *name* in the environment is changed to *value*.

RETURN VALUE

The **putenv()** function returns zero on success, or -1 if an error occurs.

ERRORS

ENOMEM
Insufficient space to allocate new environment.

CONFORMING TO

SVID 3, POSIX, BSD 4.3

SEE ALSO

getenv(3), **setenv**(3), **unsetenv**(3), **environ**(5)

NAME

putpwent - write a password file entry

SYNOPSIS

```
#include <pwd.h>
#include <stdio.h>
#include <sys/types.h>

int putpwent(const struct passwd *p, FILE *stream));
```

DESCRIPTION

The **putpwent()** function writes a password entry from the structure **p** in the file associated with *stream*.

The *passwd* structure is defined in <*pwd.h*> as follows:

```
struct passwd {
        char    *pw_name;       /* user name */
        char    *pw_passwd;     /* user password */
        uid_t   pw_uid;         /* user id */
        gid_t   pw_gid;         /* group id */
        char    *pw_gecos;      /* real name */
        char    *pw_dir;        /* home directory */
        char    *pw_shell;      /* shell program */
};
```

RETURN VALUE

The **putpwent()** function returns 0 on success, or -1 if an error occurs.

ERRORS

EINVAL
Invalid (NULL) argument given.

CONFORMING TO

SVID 3

SEE ALSO

fgetpwent(3), **getpwent**(3), **setpwent**(3), **endpwent**(3), **getpwnam**(3), **getpwuid**(3),

NAME

fputc, fputs, putc, putchar, puts - output of characters and strings

SYNOPSIS

```
#include <stdio.h>

int fputc(int c, FILE *stream));
int fputs(const char *s, FILE *stream));
int putc(int c, FILE *stream));
int putchar(int c);
int puts(const char *s);
int ungetc(int c, FILE *stream));
```

DESCRIPTION

fputc() writes the character **c**, cast to an **unsigned char**, to *stream*.

fputs() writes the string **s** to *stream*, without its trailing **'\0'**.

putc() is equivalent to **fputc()** except that it may be implemented as a macro which evaluates *stream* more than once.

putchar(c); is equivalent to **putc(c,***stdout***)**.

puts() writes the string **s** and a trailing newline to *stdout*.

Calls to the functions described here can be mixed with each other and with calls to other output functions from the **stdio** library for the same output stream.

RETURN VALUES

fputc(), **putc()** and **putchar()** return the character written as an **unsigned char** cast to an **int** or **EOF** on error.

puts() and **fputs()** return a non - negative number on success, or **EOF** on error.

CONFORMING TO

ANSI - C, POSIX.1

BUGS

It is not advisable to mix calls to output functions from the **stdio** library with low - level calls to **write()** for the file descriptor associated with the same output stream; the results will be undefined and very probably not what you want.

SEE ALSO

write(2), **fopen**(3), **fwrite**(3), **gets**(3), **fseek**(3), **ferror**(3)

NAME

qsort - sorts an array

SYNOPSIS

```
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size
     int (*compar)(const void *, const void *))
```

DESCRIPTION

The **qsort()** function sorts an array with *nmemb* elements of size *size*. The *base* argument points to the start of the array.

The contents of the array are sorted in ascending order according to a comparison function pointed to by *compar*, which is called with two arguments that point to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

RETURN VALUE

The **qsort()** function returns no value.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

sort(1)

NAME

raise - send a signal to the current process

SYNOPSIS

```
#include <signal.h>

int raise (int sig));
```

DESCRIPTION

The **raise** function sends a signal to the current process. It is equivalent to

kill(getpid(),*sig*))

RETURN VALUE

0 on success, nonzero for failure.

CONFORMING TO

ANSI-C

SEE ALSO

kill(2), **signal**(2), **getpid**(2)

NAME

rand, srand - random number generator.

SYNOPSIS

```
#include <stdlib.h>

int rand(void);

void srand(unsigned int seed));
```

DESCRIPTION

The **rand()** function returns a pseudo-random integer between 0 and **RAND_MAX**.

The **srand()** function sets its argument as the seed for a new sequence of pseudo-random integers to be returned by **rand()**. These sequences are repeatable by calling **srand()** with the same seed value.

If no seed value is provided, the **rand()** function is automatically seeded with a value of 1.

RETURN VALUE

The **rand()** function returns a value between 0 and RAND_MAX. The **srand()** returns no value.

NOTES

The versions of **rand()** and **srand()** in the Linux C Library use the same random number generator as **random()** and **srandom()**, so the lower-order bits should be as random as the higher-order bits. However, on older **rand()** implementations, the lower-order bits are much less random than the higher-order bits.

In *Numerical Recipes in* **C**: *The Art* (William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling; New York: Cambridge University Press, 1990 (1st ed, p. 207)), the following comments are made:

> "If you want to generate a random integer between 1 and 10, you should always do it by
>
> ```
> j=1+(int) (10.0*rand()/(RAND_MAX+1.0));
> ```
>
> and never by anything resembling
>
> ```
> j=1+((int) (1000000.0*rand()) % 10);
> ```
>
> (which uses lower-order bits)."

Random-number generation is a complex topic. The *Numerical Recipes in* **C** book (see reference above) provides an excellent discussion of practical random-number generation issues in Chapter 7 (Random Numbers).

For a more theoretical discussion which also covers many practical issues in depth, please see Chapter 3 (Random Numbers) in Donald E. Knuth's *The Art of Computer Programming*, volume 2 (Seminumerical Algorithms), 2nd ed.; Reading, Massachusetts: Addison-Wesley Publishing Company, 1981.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

random(3), **srandom**(3), **initstate**(3),

TOC

NAME

random, srandom, initstate, setstate - random number generator.

SYNOPSIS

```
#include <stdlib.h>

long int random(void);
void srandom(unsigned int seed));
char *initstate(unsigned int seed, char *state, int n
char *setstate(char *state));
```

DESCRIPTION

The **random()** function uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to **RAND_MAX**. The period of this random number generator is very large, approximately 16*((2**31)-1).

The **srandom()** function sets its argument as the seed for a new sequence of pseudo-random integers to be returned by **random()**. These sequences are repeatable by calling **srandom()** with the same seed value. If no seed value is provided, the **random()** function is automatically seeded with a value of 1.

The **initstate()** function allows a state array *state* to be initialized for use by **random()**. The size of the state array **n** is used by **initstate()** to decide how sophisticated a random number generator it should use - the larger the state

array, the better the random numbers will be. *seed* is the seed for the initialization, which specifies a starting point for the random number sequence, and provides for restarting at the same point.

The **setstate()** function changes the state array used by the **random()** function. The state array *state* is used for random number generation until the next call to **initstate()** or **setstate()**. *state* must first have been initialized using **initstate()**.

RETURN VALUE

The **random()** function returns a value between 0 and RAND_MAX. The **srandom()** function returns no value. The **initstate()** and **setstate()** functions return a pointer to the previous state array.

ERRORS

EINVAL
A state array of less than 8 bytes was specified to **initstate()**.

NOTES

Current "optimal" values for the size of the state array **n** are 8, 32, 64, 128, and 256 bytes; other amounts will be rounded down to the nearest known amount. Using less than 8 bytes will cause an error.

CONFORMING TO

TOC

BSD 4.3

SEE ALSO

rand(3), **srand**(3)

TOC

returning -1 if the host does not exist. Otherwise is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port If the connection succeeds, a socket in the Internet domain of type is returned to the caller, and given to the remote command as and If is non-zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as being signal numbers, to be forwarded to the process group of the command. If is 0, then the (unit 2 of the remote command) will be made the same as the and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data. The protocol is described in detail in The function is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by and several other functions. Privileged Internet ports are those in the range 0 to 1023. Only the super-user is allowed to bind an address of this sort to a socket. The and functions take a remote host's IP address or name, respectively, two user names and a flag indicating whether the local user's name is that of the super-user. Then, if the user is the super-user, it checks the file. If that lookup is not done, or is unsuccessful, the in the local user's home directory is checked to see if the request for service is allowed. If this file does not exist, is not a regular file, is owned by anyone other than the user or the super-user, or is writeable by anyone other than the owner, the check automatically fails. Zero is returned if the machine name is listed in the file, or the host and remote user name are found in the file; otherwise and return -1. If the local domain (as obtained from is the same as the remote domain, only the machine name need be specified. If the IP address of the remote host is known, should be used in preference to as it does not require trusting the DNS server for the remote host's domain. The function returns a valid socket descriptor on success. It returns -1 on error and prints a diagnostic message on the standard error. The function returns a valid, bound socket descriptor on success. It returns -1 on error with the global value set according to the reason for failure. The error code is overloaded to mean ``All network ports in use.'' These functions appeared in

NAME

re_comp, re_exec - BSD regex functions

SYNOPSIS

```
#include <regex.h>

char *re_comp(char *regex));
int re_exec(char *string));
```

DESCRIPTION

re_comp is used to compile the null-terminated regular expression pointed to by *regex*. The compiled pattern occupies a static area, the pattern buffer, which is overwritten by subsequent use of **re_comp**. If *regex* is **NULL**, no operation is performed and the pattern buffer's contents are not altered.

re_exec is used to assess whether the null-terminated string pointed to by *string* matches the previously compiled *regex*.

RETURN VALUE

re_comp returns **NULL** on successful compilation of *regex* otherwise it returns a pointer to an appropriate error message.

re_exec returns 1 for a successful match, zero for failure.

CONFORMING TO

BSD 4.3

SEE ALSO

regex(7), **GNU regex manual**

NAME

readdir - read a directory

SYNOPSIS

```
#include <sys/types.h>

#include <dirent.h>

struct dirent *readdir(DIR *dir);
```

DESCRIPTION

The **readdir()** function returns a pointer to a dirent structure representing the next directory entry in the directory stream pointed to be *dir*. It returns NULL on reaching the end-of-file or if an error occurred.

The data returned by **readdir()** is overwritten by subsequent calls to **readdir()** for the same directory stream.

According to POSIX, the *dirent* structure contains a field *char d_name*[] of unspecified size, with at most **NAME_MAX** characters preceding the terminating null character. Use of other fields will harm the portability of your programs.

RETURN VALUE

The **readdir()** function returns a pointer to a dirent structure, or NULL if an error occurs or end-of-file is reached.

ERRORS

EBADF
Invalid directory stream descriptor *dir*.

CONFORMING TO

SVID 3, POSIX, BSD 4.3

SEE ALSO

read(2), **opendir**(3), **closedir**(3), **seekdir**(3), **telldir**(3), **scandir**(3)

NAME

readv, writev - read or write data into multiple buffers

SYNOPSIS

```
#include <sys/uio.h>

int readv(int filedes, const struct iovec *vector,
          size_t count));

int writev(int filedes, const struct iovec *vector,
          size_t count));
```

DESCRIPTION

The **readv()** function reads *count* blocks from the file associated with the file descriptor *filedes* into the multiple buffers described by *vector*.

The **writev()** function writes at most *count* blocks described by *vector* to the file associated with the file descriptor *filedes*.

The pointer *vector* points to a **struct iovec** defined in **<sys/uio.h>** as

```
struct iovect
{
    void *iovbase; /* Starting address  */
    size_t iov_len;    /* Number of bytes */
} ;
```

Buffers are processed in the order *vector*[**0**], *vector*[**1**], ...

vector[*count*].

The **readv()** function works just like **read**(2) except that multiple buffers are filled.

The **writev()** function works just like **write**(2) except that multiple buffers are written out.

RETURN VALUES

The **readv()** function returns the number of bytes or -1 on error; the **writev()** function returns the number of bytes written.

ERRORS

The **readv()** and **writev()** functions can fail and set *errno* to the following values:

EBADF
fd is not a valid file descriptor.

EINVAL
fd is unsuitable for reading (for **readv()**) or writing (for **writev()**).

EFAULT
buf is outside the processes' address space.

EAGAIN
Non-blocking I/O had been selected in the **open()** call, and reading or writing could not be done immediately.

EINTR
Reading or writing was interrupted before any data was transferred.

CONFORMING TO

unknown

BUGS

It is not advisable to mix calls to functions like **readv()** or **writev()**, which operate on file descriptors, with the functions from the stdio library; the results will be undefined and probably not what you want.

SEE ALSO

read(2), **write**(2)

TOC

NAME

realpath - returns the canonicalized absolute pathname.

SYNOPSIS

```
#include <sys/param.h>
#include <unistd.h>

char *realpath(char *path, char resolved_path[]);
```

DESCRIPTION

realpath expands all symbolic links and resolves references to '/./', '/../' and extra '/' characters in the null terminated string named by *path* and stores the canonicalized absolute pathname in the buffer of size **MAXPATHLEN** named by *resolved_path*. The resulting path will have no symbolic link, '/./' or '/../' components.

RETURN VALUE

If there is no error, it returns a pointer to the *resolved_path*.

Otherwise it returns a NULL pointer and places in *resolved_path* the absolute pathname of the *path* component which could not be resolved. The global variable *errno* is set to indicate the error.

ERRORS

ENOTDIR A component of the path prefix is not a directory.

EINVAL The pathname contains a character with the high-order bit set.

ENAMETOOLONG
A component of a pathname exceeded **MAXNAMLEN** characters, or an entire path name exceeded **MAXPATHLEN** characters.

ENOENT The named file does not exist.

EACCES Search permission is denied for a component of the path prefix.

ELOOP Too many symbolic links were encountered in translating the pathname.

EIO An I/O error occurred while reading from the file system.

SEE ALSO

readlink(2), **getcwd**(3)

TOC

NAME

regcomp, regexec, regerror, regfree - POSIX regex functions

SYNOPSIS

```
#include <regex.h>

int regcomp(regex_t *preg, const char *regex, int cflags));
int regexec(const regex_t *preg, const char *string,  size_t
            nmatch, regmatch_t pmatch[], int eflags));
size_t regerror(int  errcode,  const  regex_t  *preg,   char
                *errbuf, size_t errbuf_size));
void regfree(regex_t *preg));
```

POSIX REGEX COMPILING

regcomp is used to compile a regular expression into a form that is suitable for subsequent **regexec** searches.

regcomp is supplied with *preg*, a pointer to a pattern buffer storage area; *regex*, a pointer to the null-terminated string and *cflags*, flags used to determine the type of compilation.

All regular expression searching must be done via a compiled pattern buffer, thus **regexec** must always be supplied with the address of a **regcomp** initialised pattern buffer.

cflags may be the bitwise-**or** of one or more of the following:

REG_EXTENDED
Use **POSIX** Extended Regular Expression syntax when interpreting *regex*. If not set, **POSIX** Basic Regular

Expression syntax is used.

REG_ICASE
Do not differentiate case. Subsequent **regexec** searches using this pattern buffer will be case insensitive.

REG_NOSUB
Support for substring addressing of matches is not required. The *nmatch* and *pmatch* parameters to **regexec** are ignored if the pattern buffer supplied was compiled with this flag set.

REG_NEWLINE
Match-any-character operators don't match a newline.

A non-matching list (**[^...]**) not containing a newline does not match a newline.

Match-beginning-of-line operator (**^**) matches the empty string immediately after a newline, regardless of whether *eflags*, the execution flags of **regexec**, contains **REG_NOTBOL**.

Match-end-of-line operator (**$**) matches the empty string immediately before a newline, regardless of whether *eflags* contains **REG_NOTEOL**.

POSIX REGEX MATCHING

regexec is used to match a null-terminated string against the precompiled pattern buffer, *preg*. *nmatch* and *pmatch* are used to provide information regarding the location of any matches. *eflags* may be the bitwise-**or** of one or both of **REG_NOTBOL** and **REG_NOTEOL** which cause changes in matching behaviour described below.

REG_NOTBOL
The match-beginning-of-line operator always fails to match (but see the compilation flag **REG_NEWLINE** above) This flag may be used when different portions of a string are passed to **regexec** and the beginning of the string should not be interpreted as the beginning of

the line.

REG_NOTEOL
The match-end-of-line operator always fails to match (but see the compilation flag **REG_NEWLINE** above)

BYTE OFFSETS

Unless **REG_NOSUB** was set for the compilation of the pattern buffer, it is possible to obtain substring match addressing information. *pmatch* must be dimensioned to have at least *nmatch* elements. These are filled in by **regexec** with substring match addresses. Any unused structure elements will contain the value -1.

The **regmatch_t** structure which is the type of *pmatch* is defined in *regex.h*.

```
typedef struct
{
  regoff_t rm_so;
  regoff_t rm_eo;
} regmatch_t;
```

Each *rm_so* element that is not -1 indicates the start offset of the next largest substring match within the string. The relative *rm_eo* element indicates the end offset of the match.

POSIX ERROR REPORTING

regerror is used to turn the error codes that can be returned by both **regcomp** and **regexec** into error message strings.
regerror is passed the error code, *errcode*, the pattern buffer, *preg*, a pointer to a character string buffer, *errbuf*, and the size of the string buffer, *errbuf_size*. It returns the size of the *errbuf* required to contain the null-terminated error message string. If both *errbuf* and *errbuf_size* are non-zero, *errbuf* is filled in with the first *errbuf_size* - **1** characters of the error message and a terminating null.

POSIX PATTERN BUFFER FREEING

Supplying **regfree** with a precompiled pattern buffer, *preg* will free the memory allocated to the pattern buffer by the compiling process, **regcomp**.

RETURN VALUE

regcomp returns zero for a successful compilation or an error code for failure.

regexec returns zero for a successful match or **REG_NOMATCH** for failure.

ERRORS

The following errors can be returned by **regcomp**:

REG_BADRPT
Invalid use of repetition operators such as using `*' as the first character.

REG_BADBR
Invalid use of back reference operator.

REG_EBRACE
Un-matched brace interval operators.

REG_EBRACK
Un-matched bracket list operators.

REG_ERANGE
Invalid use of the range operator, eg. the ending point of the range occurs prior to the starting point.

REG_ECTYPE
Unknown character class name.

REG_ECOLLATE
Invalid collating element.

REG_EPAREN
Un-matched parenthesis group operators.

REG_ESUBREG
Invalid back reference to a subexpression.

REG_EEND
Non specific error. This is not defined by POSIX.2.

REG_EESCAPE
Trailing backslash.

REG_BADPAT
Invalid use of pattern operators such as group or list.

REG_ESIZE
Compiled regular expression requires a pattern buffer larger than 64Kb. This is not defined by POSIX.2.

REG_ESPACE
The regex routines ran out of memory.

CONFORMING TO

POSIX.2

BUGS

Currently (GNU libc snapshot 980503), GNU libc does not support collating elements in regular expressions.

SEE ALSO

regex(7), **GNU regex manual**

TOC

NAME

remove - delete a name and possibly the file it refers to

SYNOPSIS

```
#include <stdio.h>

int remove(const char *pathname));
```

DESCRIPTION

remove deletes a name from the filesystem. If that name was the last link to a file and no processes have the file open the file is deleted and the space it was using is made available for reuse.

If the name was the last link to a file but any processes still have the file open the file will remain in existence until the last file descriptor referring to it is closed.

If the name referred to a symbolic link the link is removed.

If the name referred to a socket, fifo or device the name for it is removed but processes which have the object open may continue to use it.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EFAULT *pathname* points outside your accessible address space.

EACCES Write access to the directory containing *pathname* is not allowed for the process's effective uid, or one of the directories in *pathname* did not allow search (execute) permission.

EPERM The directory containing *pathname* has the sticky-bit (**S_ISVTX**) set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it.

ENAMETOOLONG
pathname was too long.

ENOENT A directory component in *pathname* does not exist or is a dangling symbolic link.

ENOTDIR A component used as a directory in *pathname* is not, in fact, a directory.

EISDIR *pathname* refers to a directory.
ENOMEM Insufficient kernel memory was available.

EROFS *pathname* refers to a file on a read-only filesystem.

CONFORMING TO

ANSI C, SVID, AT&T, POSIX, X/OPEN, BSD 4.3

BUGS

In-felicities in the protocol underlying NFS can cause the unexpected disappearance of files which are still being used.

SEE ALSO

unlink(2), **rename**(2), **open**(2), **mknod**(2), **mkfifo**(3), **link**(2),

TOC

TOC

NAME

res_init, res_query, res_search, res_querydomain, res_mkquery, res_send, dn_comp, dn_expand - resolver routines

SYNOPSIS

```
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
extern struct state _res;

int res_init(void);

int res_query(const char *dname, int class, int type
     unsigned char *answer, int anslen));

int res_search(const char *dname, int class, int type
     unsigned char *answer, int anslen));

int res_querydomain(const char *name, const char *domain,
     int class, int type, unsigned char *answer
     int anslen));

int res_mkquery(int op, const char *dname, int class
     int type, char *data, int datalen
     char *buf, int buflen));

int res_send(const char *msg, int msglen, char *answer
     int anslen));

int dn_comp(unsigned char *exp_dn, unsigned char *comp_dn,
     int length, unsigned char **dnptrs, unsigned char *exp_dn
     unsigned char **lastdnptr));

int dn_expand(unsigned char *msg, unsigned char *eomorig,
```

unsigned char *comp_dn, **unsigned char** *exp_dn,
int *length*));

DESCRIPTION

These functions make queries to and interpret the responses from Internet domain name servers.

The **res_init()** function reads the configuration files (see resolv+(8)) to get the default domain name, search order and name server address(es). If no server is given, the local host is tried. If no domain is given, that associated with the local host is used. It can be overridden with the environment variable LOCALDOMAIN. **res_init()** is normally executed by the first call to one of the other functions.

The **res_query()** function queries the name server for the fully-qualified domain name *name* of specified *type* and *class*. The reply is left in the buffer *answer* of length *anslen* supplied by the caller.

The **res_search()** function makes a query and waits for the response like **res_query()**, but in addition implements the default and search rules controlled by RES_DEFNAMES and RES_DNSRCH (see description of *_res* options below).

The **res_querydomain()** function makes a query using **res_query()** on the concatenation of *name* and *domain*.

The following functions are lower-level routines used by **res_query()**.

The **res_mkquery()** function constructs a query message in *buf* of length *buflen* for the domain name *dname*. The query type *op* is usually QUERY, but can be any of the types defined in <*arpa/nameser.h*>. *newrr* is currently unused.

The **res_send()** function sends a pre-formatted query given in *msg* of length *msglen* and returns the answer in *answer* which is of length *anslen*. It will call **res_init()**, if it has not already been called.

The **dn_comp()** function compresses the domain name *exp_dn* and stores it in the buffer *comp_dn* of length *length*. The compression uses an array of pointers *dnptrs* to previously compressed names in the current message. The first pointer points to the beginning of the message and the list ends with NULL. The limit of the array is specified by *lastdnptr*. if *dnptr* is NULL, domain names are not compressed. If *lastdnptr* is NULL, the list of labels is not updated.

The dn_expand() function expands the compressed domain name *comp_dn* to a full domain name, which is placed in the buffer *exp_dn* of size *length*. The compressed name is contained in a query or reply message, and *msg* points to the beginning of the message.

The resolver routines use global configuration and state information contained in the structure *_res*, which is defined in <*resolv.h*>. The only field that is normally manipulated by the user is *_res.options*. This field can contain the bitwise ``or'' of the following options:

RES_INIT
True if **res_init()** has been called.

RES_DEBUG
Print debugging messages.

RES_AAONLY
Accept authoritative answers only. **res_send()** continues until it fins an authoritative answer or returns an error. [Not currently implemented].

RES_USEVC
Use TCP connections for queries rather than UDP datagrams.

RES_PRIMARY
Query primary domain name server only.

RES_IGNTC
Ignore truncation errors. Don't retry with TCP. [Not currently implemented].

RES_RECURSE

Set the recursion desired bit in queries. Recursion is carried out by the domain name server, not by **res_send()**. [Enabled by default].

RES_DEFNAMES
If set, **res_search()** will append the default domain name to single component names, ie. those that do not contain a dot. [Enabled by default].

RES_STAYOPEN
Used with RES_USEVC to keep the TCP connection open between queries.

RES_DNSRCH
If set, **res_search()** will search for host names in the current domain and in parent domains. This option is used by **gethostbyname**(3). [Enabled by default].

RETURN VALUE

The **res_init()** function returns 0 on success, or -1 if an error occurs.

The **res_query()**, **res_search()**, **res_querydomain()**, **res_mkquery()** and **res_send()** functions return the length of the response, or -1 if an error occurs.

The **dn_comp()** and **dn_expand()** functions return the length of the compressed name, or -1 if an error occurs.

FILES

/etc/resolv.conf	resolver configuration file
/etc/host.conf	resolver configuration file

CONFORMING TO

BSD 4.3

SEE ALSO

gethostbyname(3), **hostname**(7), **named**(8),

TOC

NAME

rewinddir - reset directory stream

SYNOPSIS

```
#include <sys/types.h>

#include <dirent.h>

void rewinddir(DIR *dir));
```

DESCRIPTION

The **rewinddir()** function resets the position of the directory stream *dir* to the beginning of the directory.

RETURN VALUE

The **rewinddir()** function returns no value.

CONFORMING TO

SVID 3, POSIX, BSD 4.3

SEE ALSO

opendir(3), **readdir**(3), **closedir**(3), **telldir**(3), **scandir**(3)

NAME

rint - round to closest integer

SYNOPSIS

```
#include <math.h>

double rint(double x);
```

DESCRIPTION

The **rint()** function rounds **x** to an integer value according to the prevalent rounding mode. The default rounding mode is to round to the nearest integer.

RETURN VALUE

The **rint()** function returns the integer value as a floating-point number.

CONFORMING TO

BSD 4.3

SEE ALSO

abs(3), **ceil**(3), **fabs**(3),

NAME

scandir, alphasort - scan a directory for matching entries

SYNOPSIS

```
#include <dirent.h>

int scandir(const char *dir, struct dirent ***namelist,
     int (*select)(const struct dirent *),
     int (*compar)(const struct dirent **, const struct dirent **));

int alphasort(const struct dirent **a, const struct dirent **b);
```

DESCRIPTION

The **scandir()** function scans the directory *dir*, calling **select()** on each directory entry. Entries for which **select()** returns non-zero are stored in strings allocated via **malloc()**, sorted using **qsort()** with the comparison function **compar()**, and collected in array *namelist* which is allocated via **malloc()**. If **select** is NULL, all entries are selected.

The **alphasort()** function can be used as the comparison function for the **scandir()** function to sort the directory entries into alphabetical order. Its parameters are the two directory entries, **a** and **b**, to compare.

RETURN VALUE

The **scandir()** function returns the number of directory

entries selected or -1 if an error occurs.

The **alphasort()** function returns an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

ERRORS

ENOMEM
Insufficient memory to complete the operation.

CONFORMING TO

BSD 4.3

EXAMPLE

```
/* print files in current directory in reverse order */
#include <dirent.h>
main(){
    struct dirent **namelist;
    int n;

    n = scandir(".", &namelist, 0, alphasort);
    if (n < 0)
        perror("scandir");

    else
        while(n--) printf("%s\n", namelist[n]->d_name);
}
```

SEE ALSO

opendir(3), **readdir**(3), **closedir**(3), **rewinddir**(3), **telldir**(3), **seekdir**(3).


TOC

TOC

NAME

scanf, fscanf, sscanf, vscanf, vsscanf, vfscanf - input format conversion

SYNOPSIS

```
#include <stdio.h>
int scanf( const char *format, ...);
int fscanf( FILE *stream, const char *format, ...);
int sscanf( const char *str, const char *format, ...);

#include <stdarg.h>
int vscanf( const char *format, va_list ap));
int vsscanf( const char *str, const char *format, va_list ap
int vfscanf( FILE *stream, const char *format, va_list ap
```

DESCRIPTION

The **scanf** family of functions scans input according to a *format* as described below. This format may contain *conversion specifiers*; the results from such conversions, if any, are stored through the *pointer* arguments. The **scanf** function reads input from the standard input stream *stdin*, **fscanf** reads input from the stream pointer *stream*, and **sscanf** reads its input from the character string pointed to by *str*.

The **vfscanf** function is analogous to **vfprintf**(3) and reads input from the stream pointer *stream* using a variable argument list of pointers (see **stdarg**(3). The **vscanf** function scans a variable argument list from the standard input and the **vsscanf** function scans it from a string; these are analogous to the **vprintf** and **vsprintf** functions respec-


TOC

tively.

Each successive *pointer* argument must correspond properly with each successive conversion specifier (but see `suppression' below). All conversions are introduced by the % (percent sign) character. The *format* string may also contain other characters. White space (such as blanks, tabs, or newlines) in the *format* string match any amount of white space, including none, in the input. Everything else matches only itself. Scanning stops when an input character does not match such a format character. Scanning also stops when an input conversion cannot be made (see below).

CONVERSIONS

Following the % character introducing a conversion there may be a number of *flag* characters, as follows:

***** Suppresses assignment. The conversion that follows occurs as usual, but no pointer is used; the result of the conversion is simply discarded.

a Indicates that the conversion will be **s**, the needed memory space for the string will be malloc'ed and the pointer to it will be assigned to the *char* pointer variable, which does not have to be initialised before. This flag does not exist in *ANSI* **C**.

h Indicates that the conversion will be one of **dioux** or **n** and the next pointer is a pointer to a *short int* (rather than *int)*.

l Indicates either that the conversion will be one of **dioux** or **n** and the next pointer is a pointer to a *long int* (rather than *int)*, or that the conversion will be one of **efg** and the next pointer is a pointer to *double* (rather than *float)*. Specifying two **l** flags is equivalent to the **L** flag.

L Indicates that the conversion will be either **efg** and the next pointer is a pointer to *long double* or the conversion will be **dioux** and the next pointer is a

pointer to *long long*. (Note that long long is not an *ANSI* **C** type. Any program using this will not be portable to all architectures).

q equivalent to L. This flag does not exist in *ANSI* **C**.

In addition to these flags, there may be an optional maximum field width, expressed as a decimal integer, between the % and the conversion. If no width is given, a default of `infinity' is used (with one exception, below); otherwise at most this many characters are scanned in processing the conversion. Before conversion begins, most conversions skip white space; this white space is not counted against the field width.

The following conversions are available:

% Matches a literal `%'. That is, `%%' in the format string matches a single input `%' character. No conversion is done, and assignment does not occur.

d Matches an optionally signed decimal integer; the next pointer must be a pointer to *int*.

D Equivalent to **ld**; this exists only for backwards compatibility. (Note: thus only in libc4. In libc5 and glibc the %D is silently ignored, causing old programs to fail mysteriously.)

i Matches an optionally signed integer; the next pointer must be a pointer to *int*. The integer is read in base 16 if it begins with `0x' or `0X', in base 8 if it begins with `0', and in base 10 otherwise. Only characters that correspond to the base are used.

o Matches an unsigned octal integer; the next pointer must be a pointer to *unsigned int*.

u Matches an unsigned decimal integer; the next pointer must be a pointer to *unsigned int*.

x Matches an unsigned hexadecimal integer; the next pointer must be a pointer to *unsigned int*.

X Equivalent to **x**

f Matches an optionally signed floating-point number; the next pointer must be a pointer to *float*.

e Equivalent to **f**.

g Equivalent to **f**.

E Equivalent to **f**

s Matches a sequence of non-white-space characters; the next pointer must be a pointer to *char*, and the array must be large enough to accept all the sequence and the terminating **NUL** character. The input string stops at white space or at the maximum field width, whichever occurs first.

c Matches a sequence of *width* count characters (default 1); the next pointer must be a pointer to *char*, and there must be enough room for all the characters (no terminating **NUL** is added). The usual skip of leading white space is suppressed. To skip white space first, use an explicit space in the format.

[Matches a nonempty sequence of characters from the specified set of accepted characters; the next pointer must be a pointer to *char*, and there must be enough room for all the characters in the string, plus a terminating **NUL** character. The usual skip of leading white space is suppressed. The string is to be made up of characters in (or not in) a particular set; the set is defined by the characters between the open bracket **[** character and a close bracket **]** character. The set *excludes* those characters if the first character after the open bracket is a circumflex **^**. To include a close bracket in the set, make it the first character after the open bracket or the circumflex; any other position will end the set. The hyphen character **-** is also special; when placed between two other characters, it adds all intervening characters to the set. To include a hyphen, make it the last character before the final close bracket. For instance, `[^]0-9-]' means the set `everything except close bracket, zero through nine, and hyphen'. The string ends with the appearance of a character not in the (or, with a circumflex, in) set or when the field width runs out.

p Matches a pointer value (as printed by `%p' in **printf**(3); the next pointer must be a pointer to *void*.

n Nothing is expected; instead, the number of characters consumed thus far from the input is stored through the next pointer, which must be a pointer to *int*. This is *not* a conversion, although it can be suppressed with the ***** flag. The C standard says: `Execution of a %n directive does not increment the assignment count returned at the completion of execution' but the Corrigendum seems to contradict this. Probably it is wise not to make any assumptions on the effect of %n conversions on the return value.

RETURN VALUES

These functions return the number of input items assigned, which can be fewer than provided for, or even zero, in the event of a matching failure. Zero indicates that, while there was input available, no conversions were assigned; typically this is due to an invalid input character, such as an alphabetic character for a `%d' conversion. The value **EOF** is returned if an input failure occurs before any conversion such as an end-of-file occurs. If an error or end-of-file occurs after conversion has begun, the number of conversions which were successfully completed is returned.

SEE ALSO

strtol(3), **strtoul**(3), **strtod**(3),

STANDARDS

The functions **fscanf**, **scanf**, and **sscanf** conform to ANSI C3.159-1989 (``ANSI C'').

The **q** flag is the *BSD 4.4* notation for *long long*, while **ll** or the usage of **L** in integer conversions is the GNU notation.

The Linux version of these functions is based on the *GNU libio* library. Take a look at the *info* documentation of *GNU libc* (*glibc-1.08)* for a more concise description.

BUGS

All functions are fully ANSI C3.159-1989 conformant, but provide the additional flags **q** and **a** as well as an additional behaviour of the **L** and **l** flags. The latter may be considered to be a bug, as it changes the behaviour of flags defined in ANSI C3.159-1989.

Some combinations of flags defined by *ANSI* **C** are not making sense in *ANSI* **C** (e.g. **%Ld**). While they may have a well-defined behaviour on Linux, this need not to be so on other architectures. Therefore it usually is better to use flags that are not defined by *ANSI* **C** at all, i.e. use **q** instead of **L** in combination with **diouxX** conversions or **ll**.

The usage of **q** is not the same as on *BSD 4.4*, as it may be used in float conversions equivalently to **L**.

TOC

TOC

NAME

seekdir - set the position of the next readdir() call in the directory stream.

SYNOPSIS

```
#include <dirent.h>

void seekdir(DIR *dir, off_t offset));
```

DESCRIPTION

The **seekdir()** function sets the location in the directory stream from which the next **readdir()** call will start. **seekdir()** should be used with an offset returned by **telldir()**.

RETURN VALUE

The **seekdir()** function returns no value.

CONFORMING TO

BSD 4.3

SEE ALSO

lseek(2), **opendir**(3), **readdir**(3), **rewinddir**(3), telldir **(3),** scandir (3)

NAME

setbuf, setbuffer, setlinebuf, setvbuf - stream buffering operations

SYNOPSIS

```
#include <stdio.h>

int setbuf(FILE *stream, char *buf));
int setbuffer(FILE *stream, char *buf, size_tsize
int setlinebuf(FILE *stream));
int setvbuf(FILE *stream, char *buf, int mode , size_t
size));
```

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is output or input is read from any stream attached to a terminal device (typically stdin). The function **fflush**(3) may be used to force the block out early. (See **fclose**(3).) Normally all files are block buffered. When the first I/O operation occurs on a file, **malloc**(3) is called, and a buffer is obtained. If a stream refers to a terminal (as *stdout* normally does) it is line buffered. The standard error stream *stderr* is always unbuffered by default.

The **setvbuf** function may be used at any time on any open stream to change its buffer. The *mode* parameter must be one

of the following three macros:

_IONBF
unbuffered

_IOLBF
line buffered

_IOFBF
fully buffered

Except for unbuffered files, the *buf* argument should point to a buffer at least *size* bytes long; this buffer will be used instead of the current buffer. If the argument *buf* is **NULL**, only the mode is affected; a new buffer will be allocated on the next read or write operation. The *setvbuf* function may be used at any time, but can only change the mode of a stream when it is not ``active'': that is, before any I/O, or immediately after a call to **fflush**.

The other three calls are, in effect, simply aliases for calls to **setvbuf**. The **setbuf** function is exactly equivalent to the call

```
setvbuf(stream, buf, buf ? _IOFBF : _IONBF, BUFSIZ);
```

The **setbuffer** function is the same, except that the size of the buffer is up to the caller, rather than being determined by the default **BUFSIZ**. The **setlinebuf** function is exactly equivalent to the call:

```
setvbuf(stream, (char *)NULL, _IOLBF, 0);
```

SEE ALSO

fopen(3), **fclose**(3), **fflush**(3), **puts**(3), **printf**(3)

STANDARDS

The **setbuf** and **setvbuf** functions conform to ANSI C3.159-1989 (``ANSI C'').

BUGS

The **setbuffer** and **setlinebuf** functions are not portable to versions of BSD before 4.2BSD, and may not be available under Linux. On 4.2BSD and 4.3BSD systems, **setbuf** always uses a suboptimal buffer size and should be avoided.

You must make sure that both *buf* and the space it points to still exist by the time *stream* is closed, which also happens at program termination.

For example, the following is illegal:

```
#include <stdio.h>
int main()
{
    char buf[BUFSIZ];
    setbuf(stdin, buf);
    printf("Hello, world!\n");
    return 0;
}
```

TOC

NAME

setenv - change or add an environment variable

SYNOPSIS

```
#include <stdlib.h>

int setenv(const char *name, const char *value, int overwrite

void unsetenv(const char *name));
```

DESCRIPTION

The **setenv()** function adds the variable *name* to the environment with the value *value*, if *name* does not already exist. If *name* does exist in the environment, then its value is changed to *value* if *overwrite* is non-zero; if *overwrite* is zero, then the value of *name* is not changed.

The **unsetenv()** function deletes the variable *name* from the environment.

RETURN VALUE

The **setenv()** function returns zero on success, or -1 if there was insufficient space in the environment.

CONFORMING TO

BSD 4.3

SEE ALSO

getenv(3), **putenv**(3), **environ**(5)

NAME

setjmp, sigsetjmp - save stack context for non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp(jmp_buf env));
int sigsetjmp(sigjmp_buf env, int savesigs));
```

DESCRIPTION

setjmp() and **longjmp()** are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program. **setjmp()** saves the stack context/environment in *env* for later use by **longjmp()**. The stack context will be invalidated if the function which called **setjmp()** returns.

sigsetjmp() is similar to **setjmp()**. If *savesigs* is nonzero, the set of blocked signals is saved in *env* and will be restored if a **siglongjmp()** is later performed with this *env*.

RETURN VALUE

setjmp() and **sigsetjmp()** return 0 if returning directly, and non-zero when returning from **longjmp()** using the saved context.

CONFORMING TO

POSIX

NOTES

POSIX does not specify whether **setjmp** will save the signal context. If you want to save signal masks, use **sigsetjmp**.

setjmp() and **sigsetjmp** make programs hard to understand and maintain. If possible an alternative should be used.

SEE ALSO

longjmp(3), **siglongjmp**(3)

TOC

NAME

setlocale - set the current locale.

SYNOPSIS

#include <locale.h>

char *setlocale(int *category*, **const char *** *locale)*);

DESCRIPTION

The **setlocale()** function is used to set or query the program's current locale. If *locale* is the current locale is set to the portable locale.

If *locale* is the locale is set to the default locale which is selected from the environment variable **LANG**.

On startup of the main program, the portable **""** **"C"** locale is selected as default.

The argument *category* determines which functions are influenced by the new locale:

LC_ALL
for all of the locale.

LC_COLLATE
for the functions **strcoll()** and **strxfrm()**.

LC_CTYPE
for the character classification and conversion routines.

LC_MONETARY
for **localeconv()**.

LC_NUMERIC
for the decimal character.

LC_TIME
for **strftime()**.

A program may be made portable to all locales by calling **setlocale(LC_ALL, "")** after program initialization, by using the values returned from a **localeconv()** call for locale - dependent information and by using **strcoll()** or **strxfrm()** to compare strings.

RETURN VALUE

A successful call to **setlocale()** returns a string that corresponds to the locale set. This string may be allocated in static storage. The string returned is such that a subsequent call with that string and its associated category will restore that part of the process's locale. The return value is **NULL** if the request cannot be honored.

CONFORMING TO

ANSI C, POSIX.1

Linux (that is, libc) supports the portable locales In the good old days there used to be support for the European Latin-1 **"" "ISO-8859-1"** locale (e.g. in libc-4.5.21 and libc-4.6.27), and the Russian **"" "KOI-8"** (more precisely, "koi-8r") locale (e.g. in libc-4.6.27), so that having an environment variable LC_CTYPE=ISO-8859-1 sufficed to make isprint() return the right answer. These days non-English speaking Europeans have to work a bit harder, and must install actual locale files.

The **printf()** family of functions may or may not honor the current locale.

SEE ALSO

locale(1), **localedef**(1), **strcoll**(3), **localeconv**(3), **strftime**(3), **locale**(7)

TOC

NAME

siginterrupt - allow signals to interrupt system calls

SYNOPSIS

```
#include <signal.h>

int siginterrupt(int sig, int flag));
```

DESCRIPTION

The **siginterrupt()** function changes the restart behaviour when a system call is interrupted by the signal *sig*. If the *flag* argument is false (0), then system calls will be restarted if interrupted by the specified signal *sig*. This is the default behaviour in Linux. However, when a new signal handler is specified with the **signal**(2) function, the system call is interrupted by default.

If the *flags* argument is true (1) and no data has been transferred, then a system call interrupted by the signal *sig* will return -1 and the global variable *errno* will be set to **EINTR**.

If the *flags* argument is true (1) and data transfer has started, then the system call will be interrupted and will return the actual amount of data transferred.

RETURN VALUE

The **siginterrupt()** function returns 0 on success, or -1 if the signal number *sig* is invalid.

ERRORS

EINVAL
The specified signal number is invalid.

CONFORMING TO

BSD 4.3

SEE ALSO

signal(2)

TOC

NAME

sigemptyset, sigfillset, sigaddset, sigdelset, sigismember - POSIX signal set operations.

SYNOPSIS

```
#include <signal.h>

int sigemptyset(sigset_t *set));

int sigfillset(sigset_t *set));

int sigaddset(sigset_t *set, int signum));

int sigdelset(sigset_t *set, int signum));

int sigismember(const sigset_t *set, int signum));
```

DESCRIPTION

The **sigsetops**(3) functions allow the manipulation of POSIX signal sets.

sigemptyset initializes the signal set given by *set* to empty, with all signals excluded from the set.

sigfillset initializes *set* to full, including all signals.

sigaddset and **sigdelset** add and delete respectively signal *signum* from *set*.

sigismember tests whether *signum* is a member of *set*.

RETURN VALUES

sigemptyset, **sigfullset**, **sigaddset** and **sigdelset** return 0 on success and -1 on error.

sigismember returns 1 if *signum* is a member of *set*, 0 if *signum* is not a member, and -1 on error.

ERRORS

EINVAL
sig is not a valid signal.

CONFORMING TO

POSIX

SEE ALSO

sigaction(2), **sigpending**(2), **sigprocmask**(2), **sigsuspend**(2)

TOC

NAME

sin - sine function

SYNOPSIS

```
#include <math.h>

double sin(double x);
```

DESCRIPTION

The **sin()** function returns the sine of **x**, where **x** is given in radians.

RETURN VALUE

The **sin()** function returns a value between -1 and 1.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acos(3), **asin**(3), **atan**(3),

NAME

sinh - hyperbolic sine function

SYNOPSIS

```
#include <math.h>

double sinh(double x);
```

DESCRIPTION

The **sinh()** function returns the hyperbolic sine of **x**, which is defined mathematically as (exp(x) - exp(-x)) / 2.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acosh(3), **asinh**(3), **atanh**(3),

TOC

NAME

sleep - Sleep for the specified number of seconds

SYNOPSIS

```
#include <unistd.h>

unsigned int sleep(unsigned int seconds));
```

DESCRIPTION

sleep() makes the current process sleep until *seconds* seconds have elapsed or a signal arrives which is not ignored.

RETURN VALUE

Zero if the requested time has elapsed, or the number of seconds left to sleep.

CONFORMING TO

POSIX.1

BUGS

sleep() may be implemented using **SIGALRM**; mixing calls to **alarm()** and **sleep()** is a bad idea.

Using **longjmp()** from a signal handler or modifying the handling of **SIGALRM** while sleeping will cause undefined results.

SEE ALSO

signal(2), **alarm**(2)

TOC

NAME

snprintf, vsnprintf - formatted output conversion

SYNOPSIS

```
#define _GNU_SOURCE
#include <stdio.h>

int snprintf ( char *str, size_t n,
                          const char *format, ... );

#include <stdarg.h>

int vsnprintf ( char *str, size_t n,
                          const char *format, va_list ap );
```

DESCRIPTION

snprintf writes output to the string *str*, under control of the *format* string that specifies how subsequent arguments are converted for output. It is similar to **sprintf**(3), except that **n** specifies the maximum number of characters to produce. The trailing null character is counted towards this limit, so you should allocate at least **n** characters for the string *str*.

vsnprintf is the equivalent of **snprintf** with the variable argument list specified directly as for **vprintf**.

RETURN VALUE

If the output was truncated, the return value is -1, otherwise it is the number of characters stored, not including the terminating null.

EXAMPLES

Here is an example program which dynamically enlarges its output buffer.

```
/* Construct a message describing the value of a
   variable whose name is NAME and whose value is
   VALUE. */
char *
make_message (char *name, char *value)
{
  /* Guess we need no more than 100 chars of space. */
  int size = 100;
  char *buffer = (char *) xmalloc (size);
  while (1)
    {
      /* Try to print in the allocated space. */
      int nchars = snprintf (buffer, size,
                             "value of %s is %s", name, value);
      /* If that worked, return the string. */
      if (nchars > -1)
        return buffer;
      /* Else try again with twice as much space. */
      size *= 2;
      buffer = (char *) xrealloc (buffer, size);
    }
}
```

CONFORMING TO

These are GNU extensions.

SEE ALSO

printf(3), **sprintf**(3), **vsprintf**(3),

TOC

NAME

sqrt - square root function

SYNOPSIS

```
#include <math.h>

double sqrt(double x);
```

DESCRIPTION

The **sqrt()** function returns the non-negative square root of **x**. It fails and sets *errno* to EDOM, if **x** is negative.

ERRORS

EDOM **x** is negative.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

hypot(3)

TOC

NAME

stdarg - variable argument lists

SYNOPSIS

```
#include <stdarg.h>

void va_start( va_list ap, last));

void va_end( va_list ap));
```

DESCRIPTION

A function may be called with a varying number of arguments of varying types. The include file *stdarg.h* declares a type **va_list** and defines three macros for stepping through a list of arguments whose number and types are not known to the called function.

The called function must declare an object of type **va_list** which is used by the macros **va_start**, **va_arg**, and **va_end**.

The **va_start** macro initializes *ap* for subsequent use by **va_arg** and **va_end**, and must be called first.

The parameter *last* is the name of the last parameter before the variable argument list, i.e., the last parameter of which the calling function knows the type.

Because the address of this parameter is used in the **va_start** macro, it should not be declared as a register variable, or as a function or an array type.

The **va_start** macro returns no value.

The **va_arg** macro expands to an expression that has the type and value of the next argument in the call. The parameter *ap* is the **va_list ap** initialized by **va_start**. Each call to **va_arg** modifies *ap* so that the next call returns the next argument. The parameter *type* is a type name specified so that the type of a pointer to an object that has the specified type can be obtained simply by adding a * to *type*.

If there is no next argument, or if *type* is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), random errors will occur.

The first use of the **va_arg** macro after that of the **va_start** macro returns the argument after *last*. Successive invocations return the values of the remaining arguments.

The **va_end** macro handles a normal return from the function whose variable argument list was initialized by **va_start**. The **va_end** macro returns no value.

EXAMPLES

The function *foo* takes a string of format characters and prints out the argument associated with each format character based on the type.

```
     void foo(char *fmt, ...)
     {
          va_list ap;
          int d;
          char c, *p, *s;

          va_start(ap, fmt);
          while (*fmt)
               switch(*fmt++) {
               case 's':              /* string */
                    s = va_arg(ap, char *);
                    printf("string %s\n", s);
                    break;
               case 'd':              /* int */
```


```
                d = va_arg(ap, int);
                printf("int %d\n", d);
                break;
            case 'c':           /* char */
                c = va_arg(ap, char);
                printf("char %c\n", c);
                break;
            }
        va_end(ap);
    }
```

STANDARDS

The **va_start**, **va_arg**, and **va_end** macros conform to ANSI C3.159-1989 (``ANSI C'').

COMPATIBILITY

These macros are *not* compatible with the historic macros they replace. A backward compatible version can be found in the include file *varargs.h*.

BUGS

Unlike the **varargs** macros, the **stdarg** macros do not permit programmers to code a function with no fixed arguments. This problem generates work mainly when converting **varargs** code to **stdarg** code, but it also creates difficulties for variadic functions that wish to pass all of their arguments on to a function that takes a **va_list** argument, such as **vfprintf**(3).

stream is referred to as ``standard output''; and the error stream is referred to as ``standard error''. These terms are abbreviated to form the symbols used to refer to these files, namely and Each of these symbols is a macro of type pointer to FILE, and can be used with functions like or Since FILEs are a buffering wrapper around Unix file descriptors, the same underlying files may also be accessed using the raw Unix file interface, that is, the functions like and The integer file descriptors associated with the streams and are 0, 1, and 2, respectively. The preprocessor symbols STDIN_FILENO, STDOUT_FILENO, and STDERR_FILENO are defined with these values in <unistd.h>. Note that mixing use of FILEs and raw file descriptors can produce unexpected results and should generally be avoided. (For the masochistic among you: POSIX.1, section 8.2.3, describes in detail how this interaction is supposed to work.) A general rule is that file descriptors are handled in the kernel, while stdio is just a library. This means for example, that after an exec, the child inherits all open file descriptors, but all old streams have become inaccessible. Since the symbols and are specified to be macros, assigning to them is non-portable. The standard streams can be made to refer to different files with help of the library function specially introduced to make it possible to reassign and The standard streams are closed by a call to and by normal program termination. The stream is unbuffered. The stream is line-buffered when it points to a terminal. Partial lines will not appear until or is called, or a newline is printed. This can produce unexpected results, especially with debugging output. The buffering mode of the standard streams (or any other stream) can be changed using the or call. Note that in case is associated with a terminal, there may also be input buffering in the terminal driver, entirely unrelated to stdio buffering. (Indeed, normally terminal input is line buffered in the kernel.) This kernel input handling can be modified using calls like see also and The and macros conform to and this standard also stipulates that these three streams shall be open at program startup.

TOC

NAME

stdio - standard input/output library functions

SYNOPSIS

```
#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;
```

DESCRIPTION

The standard I/O library provides a simple and efficient buffered stream I/O interface. Input and output is mapped into logical data streams and the physical I/O characteristics are concealed. The functions and macros are listed below; more information is available from the individual man pages.

A stream is associated with an external file (which may be a physical device) by *opening* a file, which may involve creating a new file. Creating an existing file causes its former contents to be discarded. If a file can support positioning requests (such as a disk file, as opposed to a terminal) then a *file position indicator* associated with the stream is positioned at the start of the file (byte zero), unless the file is opened with append mode. If append mode is used, the position indicator will be placed the end-of-file. The position indicator is maintained by subsequent reads, writes and positioning requests. All input occurs as if the characters were read by successive calls to the **fgetc**(3) function; all output takes place as if all characters were read by

successive calls to the **fputc**(3) function.

A file is disassociated from a stream by *closing* the file. Output streams are flushed (any unwritten buffer contents are transferred to the host environment) before the stream is disassociated from the file. The value of a pointer to a **FILE** object is indeterminate after a file is closed (garbage).

A file may be subsequently reopened, by the same or another program execution, and its contents reclaimed or modified (if it can be repositioned at the start). If the main function returns to its original caller, or the **exit**(3) function is called, all open files are closed (hence all output streams are flushed) before program termination. Other methods of program termination, such as **abort**(3) do not bother about closing files properly.

At program startup, three text streams are predefined and need not be opened explicitly - *standard input* (for reading conventional input), - *standard output* (for writing conventional input), and *standard error* (for writing diagnostic output). These streams are abbreviated *stdin*,*stdout* and *stderr*. When opened, the standard error stream is not fully buffered; the standard input and output streams are fully buffered if and only if the streams do not to refer to an interactive device.

Output streams that refer to terminal devices are always line buffered by default; pending output to such streams is written automatically whenever an input stream that refers to a terminal device is read. In cases where a large amount of computation is done after printing part of a line on an output terminal, it is necessary to **fflush**(3) the standard output before going off and computing so that the output will appear.

The **stdio** library is a part of the library **libc** and routines are automatically loaded as needed by the compilers **cc**(1) and **pc**(1). The **SYNOPSIS** sections of the following manual pages indicate which include files are to be used, what the compiler declaration for the function looks like and which external variables are of interest.

The following are defined as macros; these names may not be re-used without first removing their current definitions


TOC

with **#undef**: **BUFSIZ**, **EOF**, **FILENAME_MAX**, **FOPEN_MAX**, **L_cuserid**, **L_ctermid**, **L_tmpnam**, **NULL**, **SEEK_END**, **SEEK_SET**, **SEE_CUR**, **TMP_MAX**, **clearerr**, **feof**, **ferror**, **fileno**, **fropen**, **fwopen**, **getc**, **getchar**, **putc**, **putchar**, **stderr**, **stdin**, **stdout**. Function versions of the macro functions **feof**, **ferror**, **clearerr**, **fileno**, **getc**, **getchar**, **putc**, and **putchar** exist and will be used if the macros definitions are explicitly removed.

SEE ALSO

open(2), **close**(2), **read**(2), **write**(2), **stdout**(3)

BUGS

The standard buffered functions do not interact well with certain other library and system functions, especially **vfork** and **abort**. This may not be the case under Linux.

STANDARDS

The **stdio** library conforms to ANSI C3.159-1989 (``ANSI C'').

LIST OF FUNCTIONS

Function
Description

clearerr
check and reset stream status

fclose
     close a stream

fdopen
     stream open functions

feof check and reset stream status

ferror
     check and reset stream status

fflush
     flush a stream

fgetc
     get next character or word from input stream

fgetline
     get a line from a stream

fgetpos
     reposition a stream

fgets
     get a line from a stream

fileno
     check and reset stream status

fopen
     stream open functions

fprintf
     formatted output conversion

fpurge
     flush a stream

fputc
     output a character or word to a stream

fputs
     output a line to a stream

fread
     binary stream input/output

freopen
     stream open functions

fropen
     open a stream

fscanf
     input format conversion

fseek
     reposition a stream

fsetpos
     reposition a stream

ftell
     reposition a stream

fwrite
     binary stream input/output

getc get next character or word from input stream

getchar
     get next character or word from input stream

gets get a line from a stream

getw get next character or word from input stream

mktemp
     make temporary file name (unique)

perror
     system error messages

printf
     formatted output conversion

putc output a character or word to a stream

putchar
     output a character or word to a stream

puts output a line to a stream

putw output a character or word to a stream

remove
     remove directory entry

rewind
     reposition a stream

scanf
     input format conversion

setbuf
     stream buffering operations

setbuffer
     stream buffering operations

setlinebuf
     stream buffering operations

setvbuf
     stream buffering operations

sprintf
     formatted output conversion

sscanf
     input format conversion

strerror
     system error messages

sys_errlist
     system error messages

sys_nerr
     system error messages

tempnam
     temporary file routines

tmpfile
     temporary file routines

tmpnam


```
     temporary file routines

ungetc
     un-get character from input stream

vfprintf
     formatted output conversion

vfscanf
     input format conversion

vprintf
     formatted output conversion

vscanf
     input format conversion

vsprintf
     formatted output conversion

vsscanf
     input format conversion
```


TOC

TOC

NAME

stpcpy - copy a string returning a pointer to its end

SYNOPSIS

```
#include <string.h>

char *stpcpy(char *dest, const char *src));
```

DESCRIPTION

The **stpcpy()** function copies the string pointed to by *src* (including the terminating `\0' character) to the array pointed to by *dest*. The strings may not overlap, and the destination string *dest* must be large enough to receive the copy.

RETURN VALUE

stpcpy() returns a pointer to the **end** of the string *dest* (that is, the address of the terminating null character) rather than the beginning.

EXAMPLE

For example, this program uses **stpcpy** to concatenate **foo** and **bar** to produce **foobar**, which it then prints.

```
#include <string.h>

int
main (void)
{
  char *to = buffer;
  to = stpcpy (to, "foo");
  to = stpcpy (to, "bar");
  printf ("%s\n", buffer);
}
```

CONFORMING TO

This function is not part of the ANSI or POSIX standards, and is not customary on Unix systems, but is not a GNU invention either. Perhaps it comes from MS-DOS.

SEE ALSO

strcpy(3), **bcopy**(3), **memccpy**(3),

NAME

strcasecmp, strncasecmp - compare two strings ignoring case

SYNOPSIS

```
#include <string.h>

int strcasecmp(const char *s1, const char *s2));

int strncasecmp(const char *s1, const char *s2, size_t n
```

DESCRIPTION

The **strcasecmp()** function compares the two strings *s1* and *s2*, ignoring the case of the characters. It returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*.

The **strncasecmp()** function is similar, except it only compares the first **n** characters of *s1*.

RETURN VALUE

The **strcasecmp()** and **strncasecmp()** functions return an integer less than, equal to, or greater than zero if *s1* (or the first **n** bytes thereof) is found, respectively, to be less than, to match, or be greater than *s2*.

CONFORMING TO

BSD 4.3

SEE ALSO

bcmp(3), **memcmp**(3), **strcmp**(3),

NAME

strcat, strncat - concatenate two strings

SYNOPSIS

```
#include <string.h>

char *strcat(char *dest, const char *src));

char *strncat(char *dest, const char *src, size_t n
```

DESCRIPTION

The **strcat()** function appends the *src* string to the *dest* string overwriting the `\0' character at the end of *dest*, and then adds a terminating `\0' character. The strings may not overlap, and the *dest* string must have enough space for the result.

The **strncat()** function is similar, except that only the first **n** characters of *src* are appended to *dest*.

RETURN VALUE

The **strcat()** and **strncat()** functions return a pointer to the resulting string *dest*.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

bcopy(3), **memccpy**(3), **memcpy**(3),

NAME

strchr, strrchr - locate character in string

SYNOPSIS

```
#include <string.h>

char *strchr(const char *s, int c);

char *strrchr(const char *s, int c);
```

DESCRIPTION

The **strchr()** function returns a pointer to the first occurrence of the character **c** in the string **s**.

The **strrchr()** function returns a pointer to the last occurrence of the character **c** in the string **s**.

RETURN VALUE

The **strchr()** and **strrchr()** functions return a pointer to the matched character or NULL if the character is not found.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

index(3), **memchr**(3), **rindex**(3), **strsep**(3), **strspn**(3), **strstr**(3),

TOC

NAME

strcmp, strncmp - compare two strings

SYNOPSIS

```
#include <string.h>

int strcmp(const char *s1, const char *s2));

int strncmp(const char *s1, const char *s2, size_t n
```

DESCRIPTION

The **strcmp()** function compares the two strings *s1* and *s2*. It returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*.

The **strncmp()** function is similar, except it only compares the first **n** characters of *s1*.

RETURN VALUE

The **strcmp()** and **strncmp()** functions return an integer less than, equal to, or greater than zero if *s1* (or the first **n** bytes thereof) is found, respectively, to be less than, to match, or be greater than *s2*.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

bcmp(3), **memcmp**(3), **strcasecmp**(3), **strcoll**(3)

NAME

strcoll - compare two strings using the current locale

SYNOPSIS

```
#include <string.h>

int strcoll(const char *s1, const char *s2));
```

DESCRIPTION

The **strcoll()** function compares the two strings *s1* and *s2*. It returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*. The comparison is based on strings interpreted as appropriate for the program's current locale for category *LC_COLLATE*. (See **setlocale**(3)).

RETURN VALUE

The **strcoll()** function returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*, when both are interpreted as appropriate for the current locale.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

NOTES

In the "*POSIX*" or "**C**" locales **strcoll()** is equivalent to **strcmp()**.

SEE ALSO

bcmp(3), **memcmp**(3), **strcasecmp**(3), **strxfrm**(3), **setlocale**(3)

TOC

NAME

strcpy, strncpy - copy a string

SYNOPSIS

```
#include <string.h>

char *strcpy(char *dest, const char *src));

char *strncpy(char *dest, const char *src, size_t n
```

DESCRIPTION

The **strcpy()** function copies the string pointed to be *src* (including the terminating `\0' character) to the array pointed to by *dest*. The strings may not overlap, and the destination string *dest* must be large enough to receive the copy.

The **strncpy()** function is similar, except that not more than **n** bytes of *src* are copied. Thus, if there is no null byte among the first **n** bytes of *src*, the result wil not be null-terminated.

In the case where the length of *src* is less than that of **n**, the remainder of *dest* will be padded with nulls.

RETURN VALUE

The **strcpy()** and **strncpy()** functions return a pointer to the destination string *dest*.

BUGS

If the destination string of a **strcpy()** is not large enough (that is, if the programmer was stupid/lazy, and failed to check the size before copying) then anything might happen. Overflowing fixed length strings is a favourite cracker technique.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

bcopy(3), **memccpy**(3), **memcpy**(3),

NAME

strdup - duplicate a string

SYNOPSIS

```
#include <string.h>

char *strdup(const char *s);
```

DESCRIPTION

The **strdup()** function returns a pointer to a new string which is a duplicate of the string **s**. Memory for the new string is obtained with **malloc**(3), and can be freed with **free**(3).

RETURN VALUE

The **strdup()** function returns a pointer to the duplicated string, or NULL if insufficient memory was available.

ERRORS

ENOMEM
Insufficient memory available to allocate duplicate

string.

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

calloc(3), **malloc**(3), **realloc**(3),

NAME

strerror - return string describing error code

SYNOPSIS

```
#include <string.h>

char *strerror(int errnum));
```

DESCRIPTION

The **strerror()** function returns a string describing the error code passed in the argument *errnum*. The string can only be used until the next call to **strerror()**.

RETURN VALUE

The **strerror()** function returns the appropriate description string, or an unknown error message if the error code is unknown.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

errno(3), **perror**(3), **strsignal**(3)

NAME

strfry - randomize a string

SYNOPSIS

```
#include <string.h>

char *strfry(char *string));
```

DESCRIPTION

The **strfry()** function randomizes the contents of *string* by using **rand**(3) to randomly swap characters in the string. The result is an anagram of *string*.

RETURN VALUE

The **strfry()** functions returns a pointer to the randomized string.

CONFORMING TO

The **strfry()** function is unique to the Linux C Library and GNU C Library.

SEE ALSO

memfrob(3)

TOC

NAME

strftime - format date and time

SYNOPSIS

```
#include <time.h>

size_t strftime(char *s, size_t max, const char *format
                const struct tm *tm));
```

DESCRIPTION

The **strftime()** function formats the broken-down time *tm* according to the format specification *format* and places the result in the character array **s** of size *max*.

Ordinary characters placed in the format string are copied to **s** without conversion. Conversion specifiers are introduced by a `%' character, and are replaced in **s** as follows:

%a The abbreviated weekday name according to the current locale.

%A The full weekday name according to the current locale.

%b The abbreviated month name according to the current locale.

%B The full month name according to the current locale.

%c The preferred date and time representation for the current locale.

%d The day of the month as a decimal number (range 01 to 31).

%H The hour as a decimal number using a 24-hour clock (range 00 to 23).

%I The hour as a decimal number using a 12-hour clock (range 01 to 12).

%j The day of the year as a decimal number (range 001 to 366).

%m The month as a decimal number (range 01 to 12).

%M The minute as a decimal number.

%p Either `am' or `pm' according to the given time value, or the corresponding strings for the current locale.

%S The second as a decimal number.

%U The week number of the current year as a decimal number, starting with the first Sunday as the first day of the first week.

%W The week number of the current year as a decimal number, starting with the first Monday as the first day of the first week.

%w The day of the week as a decimal, Sunday being 0.

%x The preferred date representation for the current locale without the time.

%X The preferred time representation for the current locale without the date.

%y The year as a decimal number without a century (range 00 to 99).

%Y The year as a decimal number including the century.

%Z The time zone or name or abbreviation.

%% A literal `%' character.

The broken-down time structure *tm* is defined in <*time.h*> as follows:

```
struct tm
{
        int     tm_sec;         /* seconds */
        int     tm_min;         /* minutes */
        int     tm_hour;        /* hours */
        int     tm_mday;        /* day of the month */
        int     tm_mon;         /* month */
        int     tm_year;        /* year */
        int     tm_wday;        /* day of the week */
        int     tm_yday;        /* day in the year */
        int     tm_isdst;       /* daylight saving time */
};
```

The members of the *tm* structure are:

tm_sec
The number of seconds after the minute, normally in the range 0 to 59, but can be up to 61 to allow for leap seconds.

tm_min
The number of minutes after the hour, in the range 0 to 59.

tm_hour
The number of hours past midnight, in the range 0 to 23.

tm_mday
The day of the month, in the range 1 to 31.

tm_mon
The number of months since January, in the range 0 to 11.

tm_year
The number of years since 1900.

tm_wday
The number of days since Sunday, in the range 0 to 6.

tm_yday
The number of days since January 1, in the range 0 to

365.

tm_isdst
A flag that indicates whether daylight saving time is in effect at the time described. The value is positive if daylight saving time is in effect, zero if it is not, and negative if the information is not available.

RETURN VALUE

The **strftime()** function returns the number of characters placed in the array **s**, not including the terminating NULL character, provided the string, including the terminating NULL, fits. Otherwise, it returns 0, and the contents of the array is undefined. (Thus at least since libc 4.4.4; very old versions of libc, such as libc 4.4.1, would return *max* if the array was too small.)

Note that the return value 0 does not necessarily indicate an error; for example, in many locales %p yields an empty string.

CONFORMING TO

ANSI C, SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

date(1), **time**(2), **ctime**(3),

TOC

TOC

NAME

strcasecmp, strcat, strchr, strcmp, strcoll, strcpy, strcspn, strdup, strfry, strlen, strncat, strncmp, strncpy, strncasecmp, strpbrk, strrchr, strsep, strspn, strstr, strtok, strxfrm, index, rindex - string operations

SYNOPSIS

```
#include <string.h>

int strcasecmp(const char *s1, const char *s2));

char *strcat(char *dest, const char *src));

char *strchr(const char *s, int c);

int strcmp(const char *s1, const char *s2));

int strcoll(const char *s1, const char *s2));

char *strcpy(char *dest, const char *src));

size_t strcspn(const char *s, const char *reject));

char *strdup(const char *s);

char *strfry(char *string));

size_t strlen(const char *s);

char *strncat(char *dest, const char *src, size_t n

int strncmp(const char *s1, const char *s2, size_t n

char *strncpy(char *dest, const char *src, size_t n
```


```
int strncasecmp(const char *s1, const char *s2, size_t n

char *strpbrk(const char *s, const char *accept));

char *strrchr(const char *s, int c);

char *strsep(char **stringp, const char *delim));

size_t strspn(const char *s, const char *accept));

char *strstr(const char *haystack, const char *needle));

char *strtok(char *s, const char *delim));

size_t strxfrm(char *dest, const char *src, size_t n

char *index(const char *s, int c);
char *rindex(const char *s, int c);
```

DESCRIPTION

The string functions perform string operations on NULL-terminated strings. See the individual man pages for descriptions of each function.

SEE ALSO

index(3), **rindex**(3), **strcasecmp**(3), **strchr**(3), **strcmp**(3), **strcoll**(3), **strcspn**(3), **strdup**(3), **strfry**(3), **strncat**(3), **strncmp**(3), **strncpy**(3), **strpbrk**(3), **strrchr**(3), **strsep**(3), **strstr**(3), **strtok**(3), **strxfrm**(3)

TOC

NAME

strlen - calculate the length of a string

SYNOPSIS

```
#include <string.h>

size_t strlen(const char *s);
```

DESCRIPTION

The **strlen()** function calculates the length of the string **s**, not including the terminating `\0' character.

RETURN VALUE

The **strlen()** function returns the number of characters in **s**.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

string(3)

NAME

strpbrk - search a string for any of a set of characters

SYNOPSIS

```
#include <string.h>

char *strpbrk(const char *s, const char *accept));
```

DESCRIPTION

The **strpbrk()** function locates the first occurrence in the string **s** of any of the characters in the string *accept*.

RETURN VALUE

The **strpbrk()** function returns a pointer to the character in **s** that matches one of the characters in *accept*.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

index(3), **memchr**(3), **rindex**(3), **strsep**(3), **strspn**(3), **strstr**(3),

TOC

NAME

strptime - convert a string representation of time to a time tm structure

SYNOPSIS

#include <time.h>

char *strptime(char ***buf*, const char ***format***, const struct tm ***tm**));

DESCRIPTION

strptime() is the complementary function to **strftime()** and converts the character string pointed to by *buf* to a time value, which is stored in the **tm** structure pointed to by *tm*, using the format specified by *format*. *format* is a character string that consists of field descriptors and text characters, reminiscent of **scanf**(3). Each field descriptor consists of a % character followd by another character that specifies the replacement for the field descriptor. All other characters are copied from *format* into the result. The following field descriptors are supported:

%% same as %

%a
%A day of week, using locale's weekday names; either the abbreviated or full name may be specified

%b
%B
%h month, using locale's month names; either the

abbreviated or full name may be specified

%c date and time as %x %X

%C date and time, in locale's long-format date and time representation

%d
%e day of month (1-31; leading zeroes are permitted but not required)

%D date as %m/%d/%y

%H
%k hour (0-23; leading zeroes are permitted but not required)

%I
%l hour (0-12; leading zeroes are permitted but not required)

%j day number of year (001-366)

%m month number (1-12; leading zeroes are permitted but not required)

%M minute (0-59; leading zeroes are permitted but not required)

%p locale's equivalent of AM or PM

%r time as %I:%M:%S %p

%R time as %H:%M

%S seconds (0-61; leading zeroes are permitted but not required. Extra second allowed for leap years)

%T time as %H:%M:%S

%w weekday number (0-6) with Sunday as the first day of the week

%x date, using locale's date format

%X time, using locale's time format

%y year within century (0-99; leading zeroes are permitted but not required. Unfortunately this makes the assumption that we are stuck in the 20th century as 1900 is automatically added onto this number for the tm_year field)

%Y year, including century (for example, 1988)

Case is ignored when matching items such as month or weekday names.

The broken-down time structure *tm* is defined in <*time.h*> as follows:

```
struct tm
{
        int     tm_sec;         /* seconds */
        int     tm_min;         /* minutes */
        int     tm_hour;        /* hours */
        int     tm_mday;        /* day of the month */
        int     tm_mon;         /* month */
        int     tm_year;        /* year */
        int     tm_wday;        /* day of the week */
        int     tm_yday;        /* day in the year */
        int     tm_isdst;       /* daylight saving time */
};
```

RETURN VALUE

The **strptime()** function returns a pointer to the character following the last character in the string pointed to by *buf*

SEE ALSO

strftime(3), **time**(2), **setlocale**(3),


TOC

BUGS

The return values point to static data, whose contents are overwritten by each call.

NOTES

This function is only available in libraries newer than version 4.6.5

The function supports only those locales specified in **locale**(7)

TOC

NAME

strsep - extract token from string

SYNOPSIS

```
#include <string.h>

char *strsep(char **stringp, const char *delim));
```

DESCRIPTION

The **strsep()** function returns the next token from the string *stringp* which is delimited by *delim*. The token is terminated with a `\0' character and *stringp* is updated to point past the token.

RETURN VALUE

The **strsep()** function returns a pointer to the token, or NULL if *delim* is not found in *stringp*.

NOTES

The **strsep()** function was introduced as a replacement for **strtok()**, since the latter cannot handle empty fields.

(However, **strtok()** conforms to ANSI-C and hence is more portable.)

CONFORMING TO

BSD 4.4

SEE ALSO

index(3), **memchr**(3), **rindex**(3), **strpbrk**(3), **strspn**(3), **strstr**(3),

NAME

strsignal - return string describing signal

SYNOPSIS

```
#include <string.h>

char *strsignal(int sig));

extern const char * const sys_siglist[];
```

DESCRIPTION

The **strsignal()** function returns a string describing the signal number passed in the argument *sig*. The string can only be used until the next call to **strsignal()**.

The array *sys_siglist* holds the signal description strings indexed by signal number.

RETURN VALUE

The **strsignal()** function returns the appropriate description string, or an unknown signal message if the signal number is invalid.

SEE ALSO

psignal(3), **strerror**(3)

TOC

NAME

strspn, strcspn - search a string for a set of characters

SYNOPSIS

```
#include <string.h>

size_t strspn(const char *s, const char *accept));

size_t strcspn(const char *s, const char *reject));
```

DESCRIPTION

The **strspn()** function calculates the length of the initial segment of **s** which consists entirely of characters in *accept*.

The **strcspn()** function calculates the length of the initial segment of **s** which consists entirely of characters not in *reject*.

RETURN VALUE

The **strspn()** function returns the number of characters in the initial segment of **s** which consist only of characters from *accept*.

The **strcspn()** function returns the number of characters in the initial segment of **s** which are not in the string *reject*.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

index(3), **memchr**(3), **rindex**(3), **strpbrk**(3), **strsep**(3), **strstr**(3),

NAME

strstr - locate a substring

SYNOPSIS

```
#include <string.h>

char *strstr(const char *haystack, const char *needle));
```

DESCRIPTION

The **strstr()** function finds the first occurrence of the substring *needle* in the string *haystack*. The terminating `\0' characters are not compared.

RETURN VALUE

The **strstr()** function returns a pointer to the beginning of the substring, or NULL if the substring is not found.

BUGS

Early versions of Linux libc (like 4.5.26) would not allow an empty argument. Later versions (like 4.6.27) work correctly, and return *haystack* when *needle* is empty.

SEE ALSO

index(3), **memchr**(3), **rindex**(3), **strpbrk**(3), **strsep**(3), **strspn**(3),

NAME

strtod - convert ASCII string to double

SYNOPSIS

#include <stdlib.h>

double strtod(const char *_nptr_, char **_endptr_));

DESCRIPTION

The **strtod()** function converts the initial portion of the string pointed to by *nptr* to **double** representation.

The expected form of the string is optional leading white space as checked by *isspace*(3), an optional plus (``+'') or minus sign (``-'') followed by a sequence of digits optionally containing a decimal-point character, optionally followed by an exponent. An exponent consists of an ``E'' or ``e'', followed by an optional plus or minus sign, followed by a non-empty sequence of digits. If the locale is not "C" or "POSIX", different formats may be used.

RETURN VALUES

The **strtod** function returns the converted value, if any.

If *endptr* is not **NULL**, a pointer to the character after the last character used in the conversion is stored in the loca-

tion referenced by *endptr*.

If no conversion is performed, zero is returned and the value of *nptr* is stored in the location referenced by *endptr*.

If the correct value would cause overflow, plus or minus **HUGE_VAL** is returned (according to the sign of the value), and **ERANGE** is stored in *errno*. If the correct value would cause underflow, zero is returned and **ERANGE** is stored in *errno*.

ERRORS

ERANGE
Overflow or underflow occurred.

CONFORMING TO

ANSI C

SEE ALSO

atof(3), **atoi**(3), **atol**(3), **strtol**(3), **strtoul**(3)

NAME

strtok - extract token from string

SYNOPSIS

#include <string.h>

char *strtok(char *s, const char * *delim)*);

DESCRIPTION

A `token' is a nonempty string of characters not occurring in the string *delim*, followed by \0 or by a character occurring in *delim*.

The **strtok()** function can be used to parse the string **s** into tokens. The first call to **strtok()** should have **s** as its first argument. Subsequent calls should have the first argument set to NULL. Each call returns a pointer to the next token, or NULL when no more tokens are found.

If a token ends with a delimiter, this delimiting character is overwritten with a \0 and a pointer to the next character is saved for the next call to **strtok**. The delimiter string *delim* may be different for each call.

BUGS

Never use this function. This function modifies its first argument. The identity of the delimiting character is lost. This function cannot be used on constant strings.

RETURN VALUE

The **strtok()** function returns a pointer to the next token, or NULL if there are no more tokens.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

index(3), **memchr**(3), **rindex**(3), **strpbrk**(3), **strsep**(3), **strspn**(3),

NAME

strtol - convert a string to a long integer.

SYNOPSIS

#include <stdlib.h>

long int strtol(const char * *nptr*, **char ** ** *endptr*, **int** *base*

DESCRIPTION

The **strtol()** function converts the string in *nptr* to a long integer value according to the given *base*, which must be between 2 and 36 inclusive, or be the special value 0.

The string must begin with an arbitrary amount of white space (as determined by **isspace**(3)) followed by a single optional `+' or `-' sign. If *base* is zero or 16, the string may then include a `0x' prefix, and the number will be read in base 16; otherwise, a zero *base* is taken as 10 (decimal) unless the next character is `0', in which case it is taken as 8 (octal).

The remainder of the string is converted to a long int value in the obvious manner, stopping at the first character which is not a valid digit in the given base. (In bases above 10, the letter `A' in either upper or lower case represents 10, `B' represents 11, and so forth, with `Z' representing 35.)

If *endptr* is not NULL, **strtol()** stores the address of the first invalid character in **endptr*. If there were no digits at all, **strtol()** stores the original value of *nptr* in **endptr*. (Thus, if **nptr* is not `\0' but *\*\*endptr* is `\0'

on return, the entire string is valid.)

RETURN VALUE

The **strtol()** function returns the result of the conversion, unless the value would underflow or overflow. If an underflow occurs, **strtol()** returns LONG_MIN. If an overflow occurs, **strtol()** returns LONG_MAX. In both cases, *errno* is set to ERANGE.

ERRORS

ERANGE
The given string was out of range; the value converted has been clamped.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

atof(3), **atoi**(3), **atol**(3),

BUGS

TOC

Ignores the current locale.

TOC

NAME

strtoul - convert a string to an unsigned long integer.

SYNOPSIS

#include <stdlib.h>

unsigned long int strtoul(const char ***nptr*, **char** ****endptr*, **int** base**)**

DESCRIPTION

The **strtoul()** function converts the string in *nptr* to an unsigned long integer value according to the given *base*, which must be between 2 and 36 inclusive, or be the special value 0.

The string must begin with an arbitrary amount of white space (as determined by **isspace**(3)) followed by a single optional `+' or `-' sign. If *base* is zero or 16, the string may then include a `0x' prefix, and the number will be read in base 16; otherwise, a zero *base* is taken as 10 (decimal) unless the next character is `0', in which case it is taken as 8 (octal).

The remainder of the string is converted to an unsigned long int value in the obvious manner, stopping at the first character which is not a valid digit in the given base. (In bases above 10, the letter `A' in either upper or lower case represents 10, `B' represents 11, and so forth, with `Z' representing 35.)

If *endptr* is not NULL, **strtoul()** stores the address of the


TOC

first invalid character in **endptr*. If there were no digits at all, **strtoul()** stores the original value of *nptr* in **endptr*. (Thus, if **nptr* is not `\0' but *\*\*endptr* is `\0' on return, the entire string is valid.)

RETURN VALUE

The **strtoul()** function returns either the result of the conversion or, if there was a leading minus sign, the negation of the result of the conversion, unless the original (non-negated) value would overflow; in the latter case, **strtoul()** returns ULONG_MAX and sets the global variable *errno* to ERANGE.

ERRORS

ERANGE
The given string was out of range; the value converted has been clamped.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

SEE ALSO

atof(3), **atoi**(3), **atol**(3),

BUGS

TOC

Ignores the current locale.

TOC

NAME

strxfrm - string transformation

SYNOPSIS

#include <string.h>

size_t strxfrm(char ****dest*, const char *****src***, **size_t n**

DESCRIPTION

The **strxfrm()** function transforms the *src* string into a form such that the result of **strcmp()** on two strings that have been transformed with **strxfrm()** is the same as the result of **strcoll()** on the two strings before their transformation. The first **n** characters of the transformed string are placed in *dest*. The transformation is based on the program's current locale for category *LC_COLLATE*. (See **setlocale**(3)).

RETURN VALUE

The **strxfrm()** function returns the number of bytes required to store the transformed string in *dest* excluding the terminating `\0' character. If the value returned is **n** or more, the contents of *dest* are indeterminate.

CONFORMING TO

SVID 3, BSD 4.3, ISO 9899

NOTES

In the "*POSIX*" or "**C**" locales **strxfrm()** is equivalent to copying the string with **strncpy()**.

SEE ALSO

bcmp(3), **memcmp**(3), **strcasecmp**(3), **strcoll**(3), **setlocale**(3)

TOC

NAME

swab - swap adjacent bytes

SYNOPSIS

```
#include <string.h>

void swab(const void *from, void *to, size_t n
```

DESCRIPTION

The **swab()** function copies **n** bytes from the array pointed to by *from* to the array pointed to by *to*, exchanging adjacent even and odd bytes. This function is used to exchange data between machines that have different low/high byte ordering.

RETURN VALUE

The **swab()** function returns no value.

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

bstring(3)

NAME

sysconf - Get configuration information at runtime

SYNOPSIS

#include <unistd.h>

long sysconf(int *name*));

DESCRIPTION

sysconf() provides a way for the application to determine values for system limits or options at runtime.

The equivalent macros defined in **<unistd.h>** can only give conservative values; if an application wants to take advantage of values which may change, a call to **sysconf()** can be made, which may yield more liberal results.

For getting information about a particular file, see **fpathconf()** or **pathconf()**.

The following values are supported for *name*. First, the POSIX.1 compatible values:

_SC_ARG_MAX
The maximum length of the arguments to the **exec()** family of functions; the corresponding macro is **ARG_MAX**.

_SC_CHILD_MAX
The number of simultaneous processes per user id, the corresponding macro is **_POSIX_CHILD_MAX**.

_SC_CLK_TCK
The number of clock ticks per second; the corresponding macro is **CLK_TCK**.

_SC_STREAM_MAX
The maximum number of streams that a process can have open at any time. The corresponding POSIX macro is **STREAM_MAX**, the corresponding standard C macro is **FOPEN_MAX**.

_SC_TZNAME_MAX
The maximum number of bytes in a timezone name, the corresponding macro is **TZNAME_MAX**.

_SC_OPEN_MAX
The maximum number of files that a process can have open at any time, the corresponding macro is **_POSIX_OPEN_MAX**.

_SC_JOB_CONTROL
This indicates whether POSIX - style job control is supported, the corresponding macro is **_POSIX_JOB_CONTROL**.

_SC_SAVED_IDS
This indicates whether a process has a saved set-user-ID and a saved set-group-ID; the corresponding macro is **_POSIX_SAVED_IDS**.

_SC_VERSION
indicates the year and month the POSIX.1 standard was approved in the format **YYYYMML;the** value **199009L** indicates the most recent revision, 1990.

Next, the POSIX.2 values:

_SC_BC_BASE_MAX
indicates the maximum *obase* value accepted by the **bc**(1) utility; the corresponding macro is **BC_BASE_MAX**.

_SC_BC_DIM_MAX
indicates the maximum value of elements permitted in an array by **bc**(1); the corresponding macro is **BC_DIM_MAX**.

_SC_BC_SCALE_MAX
indicates the maximum *scale* value allowed by **bc**(1); the

corresponding macro is **BC_SCALE_MAX**.

_SC_BC_STRING_MAX
indicates the maximum length of a string accepted by **bc**(1); the corresponding macro is **BC_STRING_MAX**.

_SC_COLL_WEIGHTS_MAX
indicates the maximum numbers of weights that can be assigned to an entry of the **LC_COLLATE order** keyword in the locale definition file; the corresponding macro is **COLL_WEIGHTS_MAX**.

_SC_EXPR_NEST_MAX
is the maximum number of expressions which can be nested within parentheses by **expr**(1). The corresponding macro is **EXPR_NEST_MAX**.

_SC_LINE_MAX
The maximum length of a utility's input line length, either from standard input or from a file. This includes length for a trailing newline. The corresponding macro is **LINE_MAX**.

_SC_RE_DUP_MAX
The maximum number of repeated occurrences of a regular expression when the interval notation **\{m,n\}** is used. The value of the corresponding macro is **RE_DUP_MAX**.

_SC_2_VERSION
indicates the version of the POSIX.2 standard in the format of YYYYMML. The corresponding macro is **POSIX2_VERSION**.

_SC_2_DEV
indicates whether the POSIX.2 C language development facilities are supported. The corresponding macro is **POSIX2_C_DEV**.

_SC_2_FORT_DEV
indicates whether the POSIX.2 FORTRAN development utilities are supported. The corresponding macro is **POSIX2_FORT_RUN**.

_SC_2_FORT_RUN
indicates whether the POSIX.2 FORTRAN runtime utilities are supported. The corresponding macro is

POSIX2_FORT_RUN.

_SC_2_LOCALEDEF
indicates whether the POSIX.2 creation of locates via **localedef**(1) is supported. The corresponding macro is **_POSIX2_LOCALEDEF**.

_SC_2_SW_DEV
indicates whether the POSIX.2 software development utilities option is supported. The corresponding macro is **POSIX2_SW_DEV**.

RETURN VALUE

The value returned is the value of the system resource, 1 if a queried option is available, 0 if it is not, or -1 on error. The variable *errno* is not set.

CONFORMING TO

POSIX.1, proposed POSIX.2

BUGS

It is difficult to use **ARG_MAX** because it is not specified how much of the argument space for **exec()** is consumed by the user's environment variables.

Some returned values may be huge; they are not suitable for allocating memory.

POSIX.2 is not yet an approved standard; the information in this manpage is subject to change.

SEE ALSO

bc(1), **expr**(1), **locale**(1),

NAME

closelog, openlog, syslog - send messages to the system logger

SYNOPSIS

```
#include <syslog.h>

void openlog( char *ident, int option, int  facility

void syslog( int priority, char *format, ...)

void closelog( void )
```

DESCRIPTION

closelog() closes the descriptor being used to write to the system logger. The use of **closelog()** is optional.

openlog() opens a connection to the system logger for a program. The string pointed to by *ident* is added to each message, and is typically set to the program name. Values for *option* and *facility* are given in the next section. The use of **openlog()** is optional; It will automatically be called by **syslog()** if necessary, in which case *ident* will default to NULL.

syslog() generates a log message, which will be distributed by **syslogd**(8). *priority* is a combination of the *facility* and the *level*, values for which are given in the next section. The remaining arguments are a *format*, as in **printf**(3) and any arguments required by the *format*, except that the

two character %m will be replaced by the error message string (*strerror)* corresponding to the present value of *errno*.

PARAMETERS

This section lists the parameters used to set the values of *option*, *facility*, and *priority*.

option

The *option* argument to **openlog()** is an OR of any of these:

LOG_CONS
write directly to system console if there is an error while sending to system logger

LOG_NDELAY
open the connection immediately (normally, the connection is opened when the first message is logged)

LOG_PERROR
print to stderr as well

LOG_PID
include PID with each message

facility

The *facility* argument is used to specify what type of program is logging the message. This lets the configuration file specify that messages from different facilities will be handled differently.

LOG_AUTH
security/authorization messages (DEPRECATED Use **LOG_AUTHPRIV** instead)

LOG_AUTHPRIV
security/authorization messages (private)

LOG_CRON
clock daemon (**cron** and **at**)

LOG_DAEMON
other system daemons

LOG_KERN
kernel messages

LOG_LOCAL0 through **LOG_LOCAL7**
reserved for local use

LOG_LPR
line printer subsystem

LOG_MAIL
mail subsystem

LOG_NEWS
USENET news subsystem

LOG_SYSLOG
messages generated internally by **syslogd**

LOG_USER(default)
generic user-level messages

LOG_UUCP
UUCP subsystem

level
This determines the importance of the message. The levels are, in order of decreasing importance:
LOG_EMERG
system is unusable

LOG_ALERT
action must be taken immediately

LOG_CRIT
critical conditions

LOG_ERR
error conditions

LOG_WARNING
warning conditions

LOG_NOTICE
normal, but significant, condition

LOG_INFO
informational message

LOG_DEBUG
debug-level message

HISTORY

A **syslog** function call appeared in BSD 4.2.

SEE ALSO

logger(1), **syslog.conf**(5), **syslogd**(8)

TOC

TOC

NAME

system - execute a shell command

SYNOPSIS

```
#include <stdlib.h>

int system (const char * string));
```

DESCRIPTION

system() executes a command specified in *string* by calling **/bin/sh -c** *string*, and returns after the command has been completed. During execution of the command, **SIGCHLD** will be blocked, and **SIGINT** and **SIGQUIT** will be ignored.

RETURN VALUE

The value returned is 127 if the **execve()** call for **/bin/sh** fails, -1 if there was another error and the return code of the command otherwise.

If the value of *string* is **NULL**, **system()** returns nonzero if the shell is available, and zero if not.

system() does not affect the wait status of any other children.

CONFORMING TO

ANSI C, POSIX.2, BSD 4.3

BUGS

It is extremely unfortunate that the libc version of **system()** ignores interrupts. This makes programs that call it from a loop uninterruptable. This means that for such purposes one should not use **system()** but a private version like (warning: untested code!)

```
int my_system (const char *command) {
    int pid, status;

    if (command == 0)
        return 1;
    pid = fork();
    if (pid == -1)
        return -1;
    if (pid == 0) {
        char *argv[4];
        argv[0] = "sh";
        argv[1] = "-c";
        argv[2] = command;
        argv[3] = 0;
        execve("/bin/sh", argv, environ);
        exit(127);
    }
    do {
        if (waitpid(pid, &status, 0) == -1) {
            if (errno != EINTR)
                return -1;
        } else
            return status;
    } while(1);
}
```

Do not use **system()** from a program with suid or sgid privileges, because strange values for some environment

variables might be used to subvert system integrity. Use the **exec**(3) family of functions instead, but not **execlp**(3) or **execvp**(3). **system()** will not, in fact, work properly from programs with suid or sgid privileges on systems on which **/bin/sh** is **bash** version 2, since bash 2 drops privileges on startup. (Debian uses a modified bash which does not do this when invoked as **sh**.)

The check for the availability of **/bin/sh** is not actually performed; it is always assumed to be available. ISO C specifies the check, but POSIX.2 specifies that the return shall always be non-zero, since a system without the shell is not conforming, and it is this that is implemented.

It is possible for the shell command to return 127, so that code is not a sure indication that the **execve()** call failed; check *errno* to make sure.

SEE ALSO

sh(1), **signal**(2), **exec**(3)

NAME

tan - tangent function

SYNOPSIS

```
#include <math.h>

double tan(double x);
```

DESCRIPTION

The **tan()** function returns the tangent of **x**, where **x** is given in radians.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acos(3), **asin**(3), **atan**(3),

TOC

NAME

tanh - hyperbolic tangent function

SYNOPSIS

```
#include <math.h>

double tanh(double x);
```

DESCRIPTION

The **tanh()** function returns the hyperbolic tangent of **x**, which is defined mathematically as sinh(x) / cosh(x).

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

acosh(3), **asinh**(3), **atanh**(3),


TOC

TOC

NAME

telldir - return current location in directory stream.

SYNOPSIS

```
#include <dirent.h>

off_t telldir(DIR *dir));
```

DESCRIPTION

The **telldir()** function returns the current location associated with the directory stream *dir*.

RETURN VALUE

The **telldir()** function returns the current location in the directory stream or -1 if an error occurs.

ERRORS

EBADF
Invalid directory stream descriptor *dir*.

CONFORMING TO

BSD 4.3

SEE ALSO

opendir(3), **readdir**(3), **closedir**(3), **seekdir**(3), **scandir**(3)

NAME

tempnam - create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tempnam(const char *dir, const char *pfx));
```

DESCRIPTION

The **tempnam()** function generates a unique temporary filename using up to five characters of *pfx*, if it is not NULL. The directory to place the file is searched for in the following order:-

a) The directory specified by the environment variable *TMPDIR*, if it is writable.

b) The directory specified by the argument *dir*, if it is not NULL.

c) The directory specified by *P_tmpdir*.

d) The directory */tmp*.

The storage for the filename is allocated by **malloc()**, and so can be free'd by the function **free()**.

RETURN VALUE

The **tempnam()** function returns a pointer to the unique temporary filename, or NULL if a unique filename cannot be generated.

ERRORS

EEXIST
Unable to generate a unique filename.

CONFORMING TO

SVID 3, BSD 4.3

SEE ALSO

mktemp(3), **mkstemp**(3), **tmpnam**(3),

TOC

NAME

termios, tcgetattr, tcsetattr, tcsendbreak, tcdrain, tcflush, tcflow, cfmakeraw, cfgetospeed, cfgetispeed, cfsetispeed, cfsetospeed, tcgetpgrp, tcsetpgrp - get and set terminal attributes, line control, get and set baud rate, get and set terminal foreground process group ID

SYNOPSIS

```
#include <termios.h>
#include <unistd.h>

int tcgetattr ( int fd, struct termios *termios_p );

int tcsetattr ( int fd, int optional_actions, struct termios
*termios_p

int tcsendbreak ( int fd, int duration );

int tcdrain ( int fd );

int tcflush ( int fd, int queue_selector );

int tcflow ( int fd, int action );

int cfmakeraw ( struct termios *termios_p );

speed_t cfgetospeed ( struct termios *termios_p );

int cfsetospeed ( struct termios *termios_p, speed_t speed
);

speed_t cfgetispeed ( struct termios *termios_p );

int cfsetispeed ( struct termios *termios_p, speed_t speed
);

pid_t tcgetpgrp ( int fd );
```

int tcsetpgrp (int *fd*, **pid_t** *pgrpid* **);**

DESCRIPTION

The termios functions describe a general terminal interface that is provided to control asynchronous communications ports.

Many of the functions described here have a *termios_p* argument that is a pointer to a **termios** structure. This structure contains the following members:

```
tcflag_t c_iflag;      /* input modes */
tcflag_t c_oflag;      /* output modes */
tcflag_t c_cflag;      /* control modes */
tcflag_t c_lflag;      /* local modes */
cc_t c_cc[NCCS];       /* control chars */
```

c_iflag flag constants:

IGNBRK
ignore BREAK condition on input

BRKINT
If **IGNBRK** is not set, generate **SIGINT** on BREAK condition, else read BREAK as character \0.

IGNPAR
ignore framing errors and parity errors.

PARMRK
if **IGNPAR** is not set, prefix a character with a parity error or framing error with \377 \0. If neither **IGNPAR** nor **PARMRK** is set, read a character with a parity error or framing error as \0.

INPCK

enable input parity checking

ISTRIP
strip off eighth bit

INLCR
translate NL to CR on input

IGNCR
ignore carriage return on input

ICRNL
translate carriage return to newline on input (unless **IGNCR** is set)

IUCLC
map uppercase characters to lowercase on input

IXON enable XON/XOFF flow control on output

IXANY
enable any character to restart output

IXOFF
enable XON/XOFF flow control on input

IMAXBEL
ring bell when input queue is full

c_oflag flag constants:

OPOST
enable implementation-defined output processing

OLCUC
map lowercase characters to uppercase on output

ONLCR
map NL to CR-NL on output

OCRNL
map CR to NL on output

ONOCR
don't output CR at column 0

ONLRET

don't output CR

OFILL
send fill characters for a delay, rather than using a timed delay

OFDEL
fill character is ASCII DEL. If unset, fill character is ASCII NUL

NLDLY
newline delay mask. Values are **NL0** and **NL1**.

CRDLY
carriage return delay mask. Values are **CR0**, **CR1**, **CR2**, or **CR3**.

TABDLY
horizontal tab delay mask. Values are **TAB0**, **TAB1**, **TAB2**, **TAB3**, or **XTABS**. A value of XTABS expands tabs to spaces (with tab stops every eight columns).

BSDLY
backspace delay mask. Values are **BS0** or **BS1**.

VTDLY
vertical tab delay mask. Values are **VT0** or **VT1**.

FFDLY
form feed delay mask. Values are **FF0** or **FF1**.

c_cflag flag constants:

CSIZE
character size mask. Values are **CS5**, **CS6**, **CS7**, or **CS8**.

CSTOPB
set two stop bits, rather than one.

CREAD
enable receiver.

PARENB
enable parity generation on output and parity checking for input.

PARODD

parity for input and output is odd.

HUPCL
lower modem control lines after last process closes the device (hang up).

CLOCAL
ignore modem control lines

CIBAUD
mask for input speeds (not used).

CRTSCTS
flow control.

c_lflag flag constants:

ISIG when any of the characters INTR, QUIT, SUSP, or DSUSP are received, generate the corresponding signal.

ICANON
enable canonical mode. This enables the special characters EOF, EOL, EOL2, ERASE, KILL, REPRINT, STATUS, and WERASE, and buffers by lines.

XCASE
if **ICANON** is also set, terminal is uppercase only. Input is converted to lowercase, except for characters preceded by \. On output, uppercase characters are preceded by \ and lowercase characters are converted to uppercase.

ECHO echo input characters.

ECHOE
if **ICANON** is also set, the ERASE character erases the preceding input character, and WERASE erases the preceding word.

ECHOK
if **ICANON** is also set, the KILL character erases the current line.

ECHONL
if **ICANON** is also set, echo the NL character even if ECHO is not set.


TOC

ECHOCTL
if **ECHO** is also set, ASCII control signals other than TAB, NL, START, and STOP are echoed as ^X, where X is the character with ASCII code 0x40 greater than the control signal. For example, character 0x08 (BS) is echoed as ^H.

ECHOPRT
if **ICANON** and **IECHO** are also set, characters are printed as they are being erased.

ECHOKE
if **ICANON** is also set, KILL is echoed by erasing each character on the line, as specified by **ECHOE** and **ECHOPRT**.

FLUSHO
output is being flushed. This flag is toggled by typing the DISCARD character.

NOFLSH
disable flushing the input and output queues when generating the SIGINT and SIGQUIT signals, and flushing the input queue when generating the SIGSUSP signal.

TOSTOP
send the SIGTTOU signal to the process group of a background process which tries to write to its controlling terminal.

PENDIN
all characters in the input queue are reprinted when the next character is read. (**bash** handles typeahead this way.)

IEXTEN
enable implementation-defined input processing.

tcgetattr() gets the parameters associated with the object referred by *fd* and stores them in the **termios** structure referenced by *termios_p*. This function may be invoked from a background process; however, the terminal attributes may be subsequently changed by a foreground process.

tcsetattr() sets the parameters associated with the terminal (unless support is required from the underlying hardware


TOC

that is not available) from the **termios** structure referred to by *termios_p*. *optional_actions* specifies when the changes take effect:

TCSANOW
the change occurs immediately.

TCSADRAIN
the change occurs after all output written to *fd* has been transmitted. This function should be used when changing parameters that affect output.

TCSAFLUSH
the change occurs after all output written to the object referred by *fd* has been transmitted, and all input that has been received but not read will be discarded before the change is made.

tcsendbreak() transmits a continuous stream of zero-valued bits for a specific duration, if the terminal is using asynchronous serial data transmission. If *duration* is zero, it transmits zero-valued bits for at least 0.25 seconds, and not more that 0.5 seconds. If *duration* is not zero, it sends zero-valued bits for *duration***N** seconds, where **N** is at least 0.25, and not more than 0.5.

If the terminal is not using asynchronous serial data transmission, **tcsendbreak()** returns without taking any action.

tcdrain() waits until all output written to the object referred to by *fd* has been transmitted.

tcflush() discards data written to the object referred to by *fd* but not transmitted, or data received but not read, depending on the value of *queue_selector*:

TCIFLUSH
flushes data received but not read.

TCOFLUSH
flushes data written but not transmitted.

TCIOFLUSH
flushes both data received but not read, and data written but not transmitted.

tcflow() suspends transmission or reception of data on the object referred to by *fd*, depending on the value of *action*:

TCOOFF
suspends output.

TCOON
restarts suspended output.

TCIOFF
transmits a STOP character, which stops the terminal device from transmitting data to the system.

TCION
transmits a START character, which starts the terminal device transmitting data to the system.

The default on open of a terminal file is that neither its input nor its output is suspended.

The baud rate functions are provided for getting and setting the values of the input and output baud rates in the **termios** structure. The new values do not take effect until **tcsetattr()** is successfully called.

Setting the speed to **B0** instructs the modem to "hang up". The actual bit rate corresponding to **B38400** may be altered with **setserial**(8).

The input and output baud rates are stored in the **termios** structure.

cfmakeraw sets the terminal attributes as follows:

```
termios_p->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP
                        |INLCR|IGNCR|ICRNL|IXON);
termios_p->c_oflag &= ~OPOST;
termios_p->c_lflag &= ~(ECHO|ECHONL|ICANON|ISIG|IEXTEN);
termios_p->c_cflag &= ~(CSIZE|PARENB);
termios_p->c_cflag |= CS8;
```

cfgetospeed() returns the output baud rate stored in the **termios** structure pointed to by *termios_p*.

cfsetospeed() sets the output baud rate stored in the **termios** structure pointed to by *termios_p* to *speed*, which must be one of these constants:

B0

B50
B75
B110
B134
B150
B200
B300
B600
B1200
B1800
B2400
B4800
B9600
B19200
B38400
B57600
B115200
B230400

The zero baud rate, **B0**, is used to terminate the connection. If B0 is specified, the modem control lines shall no longer be asserted. Normally, this will disconnect the line. **CBAUDEX** is a mask for the speeds beyond those defined in POSIX.1 (57600 and above). Thus, **B57600** & **CBAUDEX** is nonzero.

cfgetispeed() returns the input baud rate stored in the **termios** structure.

cfsetispeed() sets the input baud rate stored in the **termios** structure to *speed*. If the input baud rate is set to zero, the input baud rate will be equal to the output baud rate.

tcgetpgrp() returns process group ID of foreground processing group, or -1 on error.

tcsetpgrp() sets process group ID to *pgrpid*. *pgrpid* must be the ID of a process group in the same session.

RETURN VALUES

cfgetispeed() returns the input baud rate stored in the **termios** structure.

cfgetospeed() returns the output baud rate stored in the

termios structure.

tcgetpgrp() returns process group ID of foreground process-ing group, or -1 on error.

All other functions return:

0 on success.

-1 on failure and set *errno* to indicate the error.

SEE ALSO

setserial(8)

TOC

NAME

tmpfile - create a temporary file

SYNOPSIS

```
#include <stdio.h>

FILE *tmpfile (void);
```

DESCRIPTION

The **tmpfile()** function generates a unique temporary filename using the path prefix *P_tmpdir* defined in <*stdio.h*>. The temporary file is then opened in binary read/write (w+b) mode. The file will be automatically deleted when it is closed or the program terminates.

RETURN VALUE

The **tmpfile()** function returns a stream descriptor, or NULL if a unique filename cannot be generated or the unique file cannot be opened.

ERRORS

EACCES
Search permission denied for directory in file's path prefix.

EEXIST
Unable to generate a unique filename.

EMFILE
Too many file descriptors in use by process.

ENFILE
Too many files open in system.

EROFS
Read-only filesystem.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

mktemp(3), **mkstemp**(3), **tmpnam**(3),

NAME

tmpnam - create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tmpnam(char *s);
```

DESCRIPTION

The **tmpnam()** function generates a unique temporary filename using the path prefix *P_tmpdir* defined in <*stdio.h*>. If the argument **s** is NULL, **tmpnam()** returns the address of an internal static area which holds the filename, which is overwritten by subsequent calls to **tmpnam()**. If **s** is not NULL, the filename is returned in **s**.

RETURN VALUE

The **tmpnam()** function returns a pointer to the unique temporary filename, or NULL if a unique name cannot be generated.

ERRORS

EEXIST
Unable to generate a unique filename.

CONFORMING TO

SVID 3, POSIX, BSD 4.3, ISO 9899

SEE ALSO

mktemp(3), **mkstemp**(3), **tempnam**(3),

NAME

toascii - convert character to ASCII

SYNOPSIS

```
#include <ctype.h>

int toascii (int c);
```

DESCRIPTION

toascii() converts **c** to a 7-bit **unsigned char** value that fits into the ASCII character set, by clearing the high-order bits.

RETURN VALUE

The value returned is that of the converted character.

CONFORMING TO

SVID, BSD

BUGS

Many people will be unhappy if you use this function. This function will convert accented letters into random characters.

SEE ALSO

isascii(3), **toupper**(3), **tolower**(3)

NAME

toupper, tolower - convert letter to upper or lower case

SYNOPSIS

```
#include <ctype.h>

int toupper (int c);
int tolower (int c);
```

DESCRIPTION

toupper() converts the letter **c** to upper case, if possible.

tolower() converts the letter **c** to lower case, if possible.

RETURN VALUE

The value returned is that of the converted letter, or **c** if the conversion was not possible.

CONFORMING TO

ANSI - C, BSD 4.3

TOC

BUGS

The details of what constitutes an uppercase or lowercase letter depend on the current locale. For example, the default "" **"C"** locale does not know about umlauts, so no conversion is done for them.

In some non - English locales, there are lowercase letters with no corresponding uppercase equivalent; the German sharp s is one example.

SEE ALSO

isalpha(3), **setlocale**(3), **locale**(7)

TOC

NAME

tsearch, tfind, tdelete, twalk - manage a binary tree

SYNOPSIS

```
#include <search.h>

void *tsearch (const void *key, void **rootp,
               int (*compar)(const void *, const void *));

void *tfind (const void *key, const void **rootp,
               int (*compar)(const void *, const void *));

void *tdelete (const void *key, void **rootp,
               int (*compar)(const void *, const void *));

void twalk (const void *root, void (*action)) (const void *nodep
                                  const VISIT which,
                                  const int depth)));
```

DESCRIPTION

tsearch, **tfind**, **twalk**, and **tdelete** manage a binary tree. They are generalized from Knuth (6.2.2) Algorithm T. The first field in each node of the tree is a pointer to the corresponding data item. (The calling program must store the actual data.) *compar* points to a comparison routine, which takes pointers to two items. It should return an integer which is negative, zero, or positive, depending on whether the first item is less than, equal to, or greater than the second.

tsearch searches the tree for an item. *key* points to the item to be searched for. *rootp* points to a variable which points to the root of the tree. If the tree is empty, then

the variable that *rootp* points to should be set to **NULL**. If the item is found in the tree, then **tsearch** returns a pointer to it. If it is not found, then **tsearch** adds it, and returns a pointer to the newly added item.

tfind is like **tsearch**, except that if the item is not found, then **tfind** returns **NULL**.

tdelete deletes an item from the tree. Its arguments are the same as for **tsearch**.

twalk performs depth-first, left-to-right traversal of a binary tree. *root* points to the starting node for the traversal. If that node is not the root, then only part of the tree will be visited. **twalk** calls the user function *action* each time a node is visited (that is, three times for an internal node, and once for a leaf). *action*, in turn, takes three arguments. The first is a pointer to the node being visited. The second is an integer which takes on the values **preorder**, **postorder**, and **endorder** depending on whether this is the first, second, or third visit to the internal node, or **leaf** if it is the single visit to a leaf node. (These symbols are defined in <*search.h*>.) The third argument is the depth of the node, with zero being the root.

RETURN VALUE

tsearch returns a pointer to a matching item in the tree, or to the newly added item, or **NULL** if there was insufficient memory to add the item. **tfind** returns a pointer to the item, or **NULL** if no match is found. If there are multiple elements that match the key, the element returned is unspecified.

tdelete returns a pointer to the parent of the item deleted, or **NULL** if the item was not found.

tsearch, **tfind**, and **tdelete** also return **NULL** if *rootp* was **NULL** on entry.

WARNINGS

twalk takes a pointer to the root, while the other functions take a pointer to a variable which points to the root.

twalk uses **postorder** to mean "after the left subtree, but before the right subtree". Some authorities would call this "inorder", and reserve "postorder" to mean "after both subtrees".

tdelete frees the memory required for the node in the tree. The user is responsible for freeing the memory for the corresponding data.

The example program depends on the fact that **twalk** makes no further reference to a node after calling the user function with argument "endorder" or "leaf". This works with the GNU library implementation, but is not in the SysV documentation.

EXAMPLE

The following program inserts twelve random numbers into a binary tree, then prints the numbers in order. The numbers are removed from the tree and their storage freed during the traversal.

```
#include <search.h>
#include <stdlib.h>
#include <stdio.h>

void *root=NULL;

void *xmalloc(unsigned n)
{
  void *p;
  p = malloc(n);
  if(p) return p;
  fprintf(stderr, "insufficient memory\n");
  exit(1);
}

int compare(const void *pa, const void *pb)
```


```
{
  if(*(int *)pa < *(int *)pb) return -1;
  if(*(int *)pa > *(int *)pb) return 1;
  return 0;
}

void action(const void *nodep, const VISIT which, const int depth)
{
  int *datap;
  void *val;

  switch(which)
    {
    case preorder:
      break;
    case postorder:
      datap = *(int **)nodep;
      printf("%6d\n", *datap);
      break;
    case endorder:
      datap = *(int **)nodep;
      (void)tdelete(datap, &root, compare);
      free(datap);
      break;
    case leaf:
      datap = *(int **)nodep;
      printf("%6d\n", *datap);
      val = tdelete(datap, &root, compare);
      free(datap);
      break;
    }
  return;
}

int main()
{
  int i, *ptr;
  void *val;

  for (i = 0; i < 12; i++)
    {
      ptr = (int *)xmalloc(sizeof(int));
      *ptr = rand()&0xff;
      val = tsearch((void *)ptr, &root, compare);
      if(val == NULL) exit(1);
    }
  twalk(root, action);
  return 0;
```

```
}
```

CONFORMING TO

SVID

SEE ALSO

qsort(3), **bsearch**(3), **hsearch**(3),

TOC

NAME

ttyname - return name of a terminal

SYNOPSIS

```
#include <unistd.h>

char *ttyname ( int desc );
```

DESCRIPTION

Returns a pointer to the pathname of the terminal device that is open on the file descriptor *desc*, or **NULL** on error (for example, if *desc* is not connected to a terminal).

CONFORMING TO

POSIX.1

SEE ALSO

fstat(2), **isatty**(3)

TOC

TOC

NAME

tzset - initialize time conversion information

SYNOPSIS

#include <time.h>

void tzset (void);

extern char *_tzname_[2]

DESCRIPTION

The **tzset()** function initializes the *tzname* variable from the TZ environment variable. This function is automatically called by the other time conversion functions that depend on the time zone.

If the TZ variable does not appear in the environment, the *tzname* variable is initialized with the best approximation of local wall clock time, as specified by the **tzfile**(5)-format file */usr/lib/zoneinfo/localtime*.

If the TZ variable does appear in the environment but its value is NULL or its value cannot be interpreted using any of the formats specified below, Coordinated Universal Time (UTC) is used.

The value of TZ can be one of three formats. The first format is used when there is no daylight saving time in the local time zone:

std offset

The *std* string specifies the name of the time zone and must be three or more alphabetic characters. The *offset* string immediately follows *std* and specifies the time value to be added to the local time to get Coordinated Universal Time (UTC). The *offset* is positive if the local time zone is west of the Prime Meridian and negative if it is east. The hour must be between 0 and 24, and the minutes and seconds 0 and 59.

The second format is used when there is daylight saving time:

std offset dst [*offset*],*start*[*/time*],*end*[*/time*]

There are no spaces in the specification. The initial *std* and *offset* specify the standard time zone, as described above. The *dst* string and *offset* specify the name and offset for the corresponding daylight savings time zone. If the offset is omitted, it defaults to one hour ahead of standard time.
The *start* field specifies when daylight savings time goes into effect and the *end* field specifies when the change is made back to standard time. These fields may have the following formats:

Jn
: This specifies the Julian day with **n** between 1 and 365. February 29 is never counted even in leap years.

n
: This specifies the Julian day with **n** between 1 and 365. February 29 is counted in leap years.

M*m.w.***d**
: This specifies day **d** (0 <= **d** <= 6) of week **w** (1 <= **w** <= 5) of month **m** (1 <= **m** <= 12). Week 1 is the first week in which day **d** occurs and week 5 is the last week in which day **d** occurs. Day 0 is a Sunday.

The *time* fields specify when, in the local time currently in effect, the change to the other time occurs. If omitted, the default is 02:00:00.

The third format specifies that the time zone information should be read from a file:

:[filespec]

If the file specification *filespec* is omitted, the time zone information is read from */usr/lib/zoneinfo/localtime* which is in tzfile(5) format. If *filespec* is given, it specifies another tzfile(5)-format file to read the time zone information from. If *filespec* does not begin with a `/', the file specification is relative to the system time conversion information directory */usr/lib/zoneinfo*.

FILES

/usr/lib/zoneinfo	system time zone directory
/usr/lib/zoneinfo/localtime	local time zone file
/usr/lib/zoneinfo/posixrules	rules for POSIX-style TZ's

CONFORMING TO

SVID 3, POSIX, BSD 4.3

SEE ALSO

date(1), **gettimeofday**(2), **time**(2), **getenv**(3), **tzfile**(5)

TOC

NAME

ulimit - get and set user limits

SYNOPSIS

#include <ulimit.h>

long ulimit(int *cmd*, **long** *newlimit)*);

DESCRIPTION

Warning: This routine is obsolete. The include file is no longer provided by glibc. Use getrlimit(2), setrlimit(2) and sysconf(3) instead. For the shell command **ulimit**, see **bash**(1).

The **ulimit** call will get or set some limit for the current process. The *cmd* argument can have one of the following values.

UL_GETFSIZE
Return the limit on the size of a file, in units of 512 bytes.

UL_SETFSIZE
Set the limit on the size of a file.

3 (Not implemented for Linux.) Return the maximum possible address of the data segment.

4 (Implemented but no symbolic constant provided.) Return the maximum number of files that the calling process can open.

RETURN VALUE

On success, **ulimit** returns a nonnegative value. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EPERM
A non-root process tried to increase a limit.

CONFORMING TO

SVID.

SEE ALSO

bash(1), **getrlimit**(2), **setrlimit**(2), **sysconf**(3)

NAME

none - undocumented library functions

SYNOPSIS

Undocumented library functions

DESCRIPTION

This man page mentions those library functions which are implemented in the standard libraries but not yet documented in man pages.

SOLICITATION

If you have information about these functions, please look in the source code, write a man page (using a style similar to that of the other Linux section 3 man pages), and send it to **aeb@cwi.nl** for inclusion in the next man page release.

THE LIST

des_setparity dn_skipname ecb_crypt encrypt endnetgrent endrpcent endutent fcrypt fp_nquery fp_query fp_resstat get_myaddress getnetgrent getnetname getpublickey

getrpcbyname getrpcbynumber getrpcent getrpcport getsecret-key h_errlist host2netname hostalias inet_nsap_addr inet_nsap_ntoa init_des innetgr key_decryptsession key_encryptsession key_gendes key_setsecret libc_nls_init lockf mcheck memalign mstats mtrace netname2host netname2user nlist obstack_free p_cdname p_cdnname p_class p_fqname p_option p_query p_rr p_time p_type passwd2des pmap_getmaps pmap_getport pmap_rmtcall pmap_set pmap_unset putlong putshort rcmd re_compile_fastmap re_compile_pattern re_match re_match_2 re_rx_search re_search re_search_2 re_set_registers re_set_syntax registerrpc res_send_setqhook res_send_setrhook rexec rresvport rtime ruserok ruserpass setfileno sethostfile setkey setlogmask setnetgrent setrp-cent svc_exit svc_getreq svc_getreqset svc_register svc_run svc_sendreply svc_unregister svcerr_auth svcerr_decode svcerr_noproc svcerr_noprog svcerr_progvers svcerr_systemerr svcerr_weakauth svcfd_create svcraw_create svctcp_create svcudp_bufcreate svcudp_create svcudp_enablecache syscall tell timegm tr_break tzsetwall ufc_dofinalperm ufc_doit user2netname valloc vsyslog xdecrypt xdr_accepted_reply xdr_array xdr_authdes_cred xdr_authdes_verf xdr_authunix_parms xdr_bool xdr_bytes xdr_callhdr xdr_callmsg xdr_char xdr_cryptkeyarg xdr_cryptkeyres xdr_datum xdr_des_block xdr_domainname xdr_double xdr_enum xdr_float xdr_free xdr_getcredres xdr_int xdr_keybuf xdr_keystatus xdr_long xdr_mapname xdr_netnamestr xdr_netobj xdr_opaque xdr_opaque_auth xdr_passwd xdr_peername xdr_pmap xdr_pmaplist xdr_pointer xdr_reference xdr_rejected_reply xdr_replymsg xdr_rmtcall_args xdr_rmtcallres xdr_short xdr_string xdr_u_char xdr_u_int xdr_u_long xdr_u_short xdr_union xdr_unixcred xdr_vector xdr_void xdr_wrapstring xdr_yp_buf xdr_yp_inaddr xdr_ypbind_binding xdr_ypbind_resp xdr_ypbind_resptype xdr_ypbind_setdom xdr_ypdelete_args xdr_ypmaplist xdr_ypmaplist_str xdr_yppasswd xdr_ypreq_key xdr_ypreq_nokey xdr_ypresp_all xdr_ypresp_all_seq xdr_ypresp_key_val xdr_ypresp_maplist xdr_ypresp_master xdr_ypresp_order xdr_ypresp_val xdr_ypstat xdr_ypupdate_args xdrmem_create xdrrec_create xdrrec_endofrecord xdrrec_eof xdrrec_skiprecord xdrstdio_create xencrypt xprt_register xprt_unregister yp_all yp_bind yp_first yp_get_default_domain yp_maplist yp_master yp_match yp_next yp_order yp_unbind yp_update yperr_string ypprot_err


TOC

NAME

updwtmp, logwtmp - append an entry to the wtmp file

SYNOPSIS

```
#include <utmp.h>

void updwtmp(const char *wtmp_file, const struct utmp *ut));
void logwtmp(const char *line, const char *name, const char *host
```

DESCRIPTION

updwtmp() appends the utmp structure *ut* to the wtmp file.

logwtmp() constructs an utmp structure using *line*, *name*, *host*, current time and current process id. Then it calls **updwtmp**() to append the structure to the utmp file.

AVAILABILITY

Both functions are available under glibc2, but not under glibc1 or libc. However, logwtmp() occurs in the old libbsd.

FILES

/var/log/wtmp database of past user logins

SEE ALSO

utmp(5)

NAME

usleep - suspend execution for interval of microseconds

SYNOPSIS

```
#include <unistd.h>

void usleep(unsigned long usec));
```

DESCRIPTION

The **usleep()** function suspends execution of the calling process for *usec* microseconds. The sleep may be lengthened slightly by any system activity or by the time spent processing the call.

CONFORMING TO

BSD 4.3

SEE ALSO

setitimer(2), **getitimer**(2), **sleep**(3), **alarm**(2), **select**(2)

TOC

TOC

NAME

wcstombs - convert a wide character string to a multibyte character string.

SYNOPSIS

#include <stdlib.h>

size_t wcstombs(char *s, const wchar_t *pwcs, size_t n

DESCRIPTION

The **wcstombs()** function converts a sequence of wide characters from the array *pwcs* into a sequence of multibyte characters and stores up to **n** bytes of multibyte characters in the array **s**.

RETURN VALUE

wcstombs() returns the number of bytes stored in **s** or -1 if **s** contains an invalid wide character.

CONFORMING TO

SVID 3, ISO 9899

SEE ALSO

mblen(3), **mbtowc**(3), **mbstowcs**(3),

NAME

wctomb - convert a wide character to a multibyte character.

SYNOPSIS

```
#include <stdlib.h>

int wctomb(char *s, wchar_t wchar));
```

DESCRIPTION

The **wctomb()** function converts a wide character *wchar* into a multibyte character and, if **s** is not NULL, stores the multibyte character representation in **s**.

RETURN VALUE

wctomb() returns the number of bytes in the multibyte character or -1 if the wide character is not valid.

CONFORMING TO

SVID 3, ISO 9899

SEE ALSO

mblen(3), **mbstowcs**(3), **mbtowc**(3),

Linux Man Pages Section 4

- charsets.4
- console.4
- console_codes.4
- console_ioctl.4
- fd.4
- full.4
- hd.4
- initrd.4
- intro.4
- kmem.4
- lp.4
- mem.4
- mouse.4
- null.4
- port.4
- ram.4
- random.4
- sd.4
- st.4
- tty.4
- ttys.4
- vcs.4
- vcsa.4
- wavelan.4
- zero.4

NAME

charsets - programmer's view of character sets and interna-tionalization

DESCRIPTION

Linux is an international operating system. Various of its utilities and device drivers (including the console driver) support multilingual character sets including Latin-alphabet letters with diacritical marks, accents, ligatures, and entire non-Latin alphabets including Greek, Cyrillic, Ara-bic, and Hebrew.

This manual page presents a programmer's-eye view of dif-ferent character-set standards and how they fit together on Linux. Standards discussed include ASCII, ISO 8859, KOI8-R, Unicode, ISO 2022 and ISO 4873.

ASCII

ASCII (American Standard Code For Information) is the origi-nal 7-bit character set, originally designed for American English. It is currently described by the ECMA-6 standard.

An ASCII variant replacing the American crosshatch/octothorpe/hash pound symbol with the British pound-sterling symbol is used in Great Britain; when needed, the American and British variants may be distinguished as "US ASCII" and "UK ASCII".

As Linux was written for hardware designed in the US, it natively supports US ASCII.

ISO 8859

ISO 8859 is a series of 10 8-bit character sets all of which have US ASCII in their low (7-bit) half, invisible control characters in positions 128 to 159, and 96 fixed-width graphics in positions 160-255.

Of these, the most important is ISO 8859-1 (Latin-1). It is natively supported in the Linux console driver, fairly well supported in X11R6, and is the base character set of HTML.

Console support for the other 8859 character sets is available under Linux through user-mode utilities (such as **setfont**(8)) that modify keyboard bindings and the EGA graphics table and employ the "user mapping" font table in the console driver.

Here are brief descriptions of each set:

8859-1 (Latin-1)
: Latin-1 covers most Western European languages such as Albanian, Catalan, Danish, Dutch, English, Faroese, Finnish, French, German, Galician, Irish, Icelandic, Italian, Norwegian, Portuguese, Spanish, and Swedish. The lack of the ligatures Dutch ij, French oe and old-style ,,German`` quotation marks is tolerable.

8859-2 (Latin-2)
: Latin-2 supports most Latin-written Slavic and Central European languages: Czech, German, Hungarian, Polish, Rumanian, Croatian, Slovak, and Slovene.

8859-3 (Latin-3)
: Latin-3 is popular with authors of Esperanto, Galician, Maltese, and Turkish.

8859-4 (Latin-4)
: Latin-4 introduced letters for Estonian, Latvian, and Lithuanian. It is essentially obsolete; see 8859-10 (Latin-6).

8859-5
Cyrillic letters supporting Bulgarian, Byelorussian, Macedonian, Russian, Serbian and Ukrainian. Ukrainians read the letter `ghe' with downstroke as `heh' and would need a ghe with upstroke to write a correct ghe. See the discussion of KOI8-R below.

8859-6
Supports Arabic. The 8859-6 glyph table is a fixed font of separate letter forms, but a proper display engine should combine these using the proper initial, medial, and final forms.

8859-7
Supports Modern Greek.

8859-8
Supports Hebrew.

8859-9 (Latin-5)
This is a variant of Latin-1 that replaces rarely-used Icelandic letters with Turkish ones.

8859-10 (Latin-6)
Latin 6 adds the last Inuit (Greenlandic) and Sami (Lappish) letters that were missing in Latin 4 to cover the entire Nordic area. RFC 1345 listed a preliminary and different `latin6'. Skolt Sami still needs a few more accents than these.

KOI8-R

KOI8-R is a non-ISO character set popular in Russia. The lower half is US ASCII; the upper is a Cyrillic character set somewhat better designed than ISO 8859-5.

Console support for KOI8-R is available under Linux through user-mode utilities that modify keyboard bindings and the EGA graphics table, and employ the "user mapping" font table in the console driver.

UNICODE

Unicode (ISO 10646) is a standard which aims to unambiguously represent every known glyph in every human language. Unicode's native encoding is 32-bit (older versions used 16 bits). Information on Unicode is available at <http://www.unicode.com>.

Linux represents Unicode using the 8-bit Unicode Transfer Format (UTF-8). UTF-8 is a variable length encoding of Unicode. It uses 1 byte to code 7 bits, 2 bytes for 11 bits, 3 bytes for 16 bits, 4 bytes for 21 bits, 5 bytes for 26 bits, 6 bytes for 31 bits.

Let 0,1,x stand for a zero, one, or arbitrary bit. A byte 0xxxxxxx stands for the Unicode 00000000 0xxxxxxx which codes the same symbol as the ASCII 0xxxxxxx. Thus, ASCII goes unchanged into UTF-8, and people using only ASCII do not notice any change: not in code, and not in file size.

A byte 110xxxxx is the start of a 2-byte code, and 110xxxxx 10yyyyyy is assembled into 00000xxx xxyyyyyy. A byte 1110xxxx is the start of a 3-byte code, and 1110xxxx 10yyyyyy 10zzzzzz is assembled into xxxxyyyy yyzzzzzz. (When UTF-8 is used to code the 31-bit ISO 10646 then this progression continues up to 6-byte codes.)

For ISO-8859-1 users this means that the characters with high bit set now are coded with two bytes. This tends to expand ordinary text files by one or two percent. There are no conversion problems, however, since the Unicode value of ISO-8859-1 symbols equals their ISO-8859-1 value (extended by eight leading zero bits). For Japanese users this means that the 16-bit codes now in common use will take three bytes, and extensive mapping tables are required. Many Japanese therefore prefer ISO 2022.

Note that UTF-8 is self-synchronizing: 10xxxxxx is a tail, any other byte is the head of a code. Note that the only way ASCII bytes occur in a UTF-8 stream, is as themselves. In particular, there are no embedded NULs or '/'s that form


TOC

part of some larger code.

Since ASCII, and, in particular, NUL and '/', are unchanged, the kernel does not notice that UTF-8 is being used. It does not care at all what the bytes it is handling stand for.

Rendering of Unicode data streams is typically handled through `subfont' tables which map a subset of Unicode to glyphs. Internally the kernel uses Unicode to describe the subfont loaded in video RAM. This means that in UTF-8 mode one can use a character set with 512 different symbols. This is not enough for Japanese, Chinese and Korean, but it is enough for most other purposes.

ISO 2022 AND ISO 4873

The ISO 2022 and 4873 standards describe a font-control model based on VT100 practice. This model is (partially) supported by the Linux kernel and by **xterm**(1). It is popular in Japan and Korea.

There are 4 graphic character sets, called G0, G1, G2 and G3, and one of them is the current character set for codes with high bit zero (initially G0), and one of them is the current character set for codes with high bit one (initially G1). Each graphic character set has 94 or 96 characters, and is essentially a 7-bit character set. It uses codes either 040-0177 (041-0176) or 0240-0377 (0241-0376). G0 always has size 94 and uses codes 041-0176.

Switching between character sets is done using the shift functions ^N (SO or LS1), ^O (SI or LS0), ESC n (LS2), ESC o (LS3), ESC N (SS2), ESC O (SS3), ESC ~ (LS1R), ESC } (LS2R), ESC | (LS3R). The function LS**n** makes character set G**n** the current one for codes with high bit zero. The function LS**n**R makes character set G**n** the current one for codes with high bit one. The function SS**n** makes character set G**n** (**n**=2 or 3) the current one for the next character only (regardless of the value of its high order bit).

A 94-character set is designated as G**n** character set by an

escape sequence ESC (xx (for G0), ESC) xx (for G1), ESC * xx (for G2), ESC + xx (for G3), where xx is a symbol or a pair of symbols found in the ISO 2375 International Register of Coded Character Sets. For example, ESC (@ selects the ISO 646 character set as G0, ESC (A selects the UK standard character set (with pound instead of number sign), ESC (B selects ASCII (with dollar instead of currency sign), ESC (M selects a character set for African languages, ESC (! A selects the Cuban character set, etc. etc.

A 96-character set is designated as G**n** character set by an escape sequence ESC - xx (for G1), ESC . xx (for G2) or ESC / xx (for G3). For example, ESC - G selects the Hebrew alphabet as G1.

A multibyte character set is designated as G**n** character set by an escape sequence ESC $ xx or ESC $ (xx (for G0), ESC $) xx (for G1), ESC $ * xx (for G2), ESC $ + xx (for G3). For example, ESC $ (C selects the Korean character set for G0. The Japanese character set selected by ESC $ B has a more recent version selected by ESC & @ ESC $ B.

ISO 4873 stipulates a narrower use of character sets, where G0 is fixed (always ASCII), so that G1, G2 and G3 can only be invoked for codes with the high order bit set. In particular, ^N and ^O are not used anymore, ESC (xx can be used only with xx=B, and ESC) xx, ESC * xx, ESC + xx are equivalent to ESC - xx, ESC . xx, ESC / xx, respectively.

SEE ALSO

console(4), **console_ioctl**(4), **console_codes**(4)

TOC

TOC

NAME

console - console terminal and virtual consoles

DESCRIPTION

A Linux system has up to 63 *virtual consoles* (character devices with major number 4 and minor number 1 to 63), usually called **/dev/ttyn** with 1 < **n** < 63. The current console is also addressed by **/dev/console** or **/dev/tty0**, the character device with major number 4 and minor number 0. The device files /dev/* are usually created using the script MAKEDEV, or using mknod(1), usually with mode 0622 and owner root.tty.

Before kernel version 1.1.54 the number of virtual consoles was compiled into the kernel (in tty.h: #define NR_CONSOLES 8) and could be changed by editing and recompiling. Since version 1.1.54 virtual consoles are created on the fly, as soon as they are needed.

Common ways to start a process on a console are: (a) tell init(8) (in inittab(5)) to start a getty(8) on the console; (b) ask open(1) to start a process on the console; (c) start X - it will find the first unused console, and display its output there. (There is also the ancient doshell(8).)

Common ways to switch consoles are: (a) use Alt+F**n** or Ctrl+Alt+F**n** to switch to console **n**; AltGr+F**n** might bring you to console **n**+12 [here Alt and AltGr refer to the left and right Alt keys, respectively]; (b) use Alt+RightArrow or Alt+LeftArrow to cycle through the presently allocated consoles; (c) use the program chvt(1). (The key mapping is user settable, see loadkeys(1); the above mentioned key combinations are according to the default settings.)

The command **deallocvt**(1) (formerly **disalloc**) will free the

memory taken by the screen buffers for consoles that no longer have any associated process.

PROPERTIES

Consoles carry a lot of state. I hope to document that some other time. The most important fact is that the consoles simulate vt100 terminals. In particular, a console is reset to the initial state by printing the two characters ESC c. All escape sequences can be found in **console_codes**(4).

FILES

/dev/console
*/dev/tty**

SEE ALSO

charsets(4), **console_codes**(4), **console_ioctl**(4), **mknod**(1), **tty**(4), **ttys**(4), **getty**(8), **init**(8), **chvt**(1), **open**(1), **deallocvt**(1), **loadkeys**(1), **resizecons**(8), **setfont**(8), **mapscrn**(8)

TOC

NAME

console_codes - Linux console escape and control sequences

DESCRIPTION

The Linux console implements a large subset of the VT102 and ECMA-48/ISO 6429/ANSI X3.64 terminal controls, plus certain private-mode sequences for changing the color palette, character-set mapping, etc. In the tabular descriptions below, the second column gives ECMA-48 or DEC mnemonics (the latter if prefixed with DEC) for the given function. Sequences without a mnemonic are neither ECMA-48 nor VT102.

After all the normal output processing has been done, and a stream of characters arrives at the console driver for actual printing, the first thing that happens is a translation from the code used for processing to the code used for printing.

If the console is in UTF-8 mode, then the incoming bytes are first assembled into 16-bit Unicode codes. Otherwise each byte is transformed according to the current mapping table (which translates it to a Unicode value). See the CHARACTER SETS section below for discussion.

In the normal case, the Unicode value is converted to a font index, and this is stored in video memory, so that the corresponding glyph (as found in video ROM) appears on the screen. Note that the use of Unicode (and the design of the PC hardware) allows us to use 512 different glyphs simultaneously.

If the current Unicode value is a control character, or we are currently processing an escape sequence, the value will treated specially. Instead of being turned into a font index and rendered as a glyph, it may trigger cursor move-

ment or other control functions. See the LINUX CONSOLE CONTROLS section below for discussion.

It is generally not good practice to hard-wire terminal controls into programs. Linux supports a *terminfo*(5) database of terminal capabilities. Rather than emitting console escape sequences by hand, you will almost always want to use a terminfo-aware screen library or utility such as **ncurses**(3), **tput**(1), or **reset**(1).

LINUX CONSOLE CONTROLS

This section describes all the control characters and escape sequences that invoke special functions (i.e. anything other than writing a glyph at the current cursor location) on the Linux console.

Control characters

A character is a control character if (before transformation according to the mapping table) it has one of the 14 codes 00 (NUL), 07 (BEL), 08 (BS), 09 (HT), 0a (LF), 0b (VT), 0c (FF), 0d (CR), 0e (SO), 0f (SI), 18 (CAN), 1a (SUB), 1b (ESC), 7f (DEL). One can set a `display control characters' mode (see below), and allow 07, 09, 0b, 18, 1a, 7f to be displayed as glyphs. On the other hand, in UTF-8 mode all codes 00-1f are regarded as control characters, regardless of any `display control characters' mode.

If we have a control character, it is acted upon immediately and then discarded (even in the middle of an escape sequence) and the escape sequence continues with the next character. (However, ESC starts a new escape sequence, possibly aborting a previous unfinished one, and CAN and SUB abort any escape sequence.) The recognized control characters are BEL, BS, HT, LF, VT, FF, CR, SO, SI, CAN, SUB, ESC, DEL, CSI. They do what one would expect:

BEL (0x07, ^G) beeps;

BS (0x08, ^H) backspaces one column (but not past the beginning of the line);

HT (0x09, ^I) goes to the next tab stop or to the end of the line if there is no earlier tab stop;

LF (0x0A, ^J), VT (0x0B, ^K) and FF (0x0C, ^L) all give a linefeed;

CR (0x0D, ^M) gives a carriage return;

SO (0x0E, ^N) activates the G1 character set, and if LF/NL (new line mode) is set also a carriage return;

SI (0x0F, ^O) activates the G0 character set;

CAN (0x18, ^X) and SUB (0x1A, ^Z) interrupt escape sequences;

ESC (0x1B, ^[) starts an escape sequence;

DEL (0x7F) is ignored;

CSI (0x9B) is equivalent to ESC [.

ESC- but not CSI-sequences

l l l. ESC c RIS Reset. ESC D IND Linefeed. ESC E NEL Newline. ESC H HTS Set tab stop at current column. ESC M RI Reverse linefeed. ESC Z DECID DEC private identification. The kernel returns the string ESC [? 6 c, claiming that it is a VT102. ESC 7 DECSC Save current state (cursor coordinates, attributes, character sets). ESC 8 DECRC Restore most recently saved state. ESC [CSI Control sequence introducer ESC % Start sequence selecting character set ESC % @ Select default (ISO 646 / ISO 8859-1) ESC % G Select UTF-8 ESC % 8 Select UTF-8 (obsolete) ESC # 8 DECALN DEC screen alignment test - fill screen with E's. ESC (Start sequence defining G0 character set ESC (B Select default (ISO 8859-1 mapping) ESC (0 Select vt100 graphics mapping ESC (U Select null mapping - straight to character ROM ESC (K Select user mapping - the map that is loaded by the utility **mapscrn**(8). ESC) Start sequence defining G1 (followed by one of B, 0, U, K, as above). ESC > DECPNM Set numeric keypad mode ESC

= DECPAM Set application keypad mode ESC] OSC (Should be: Operating system command) ESC] P *nrrggbb*: set palette, with parameter given in 7 hexadecimal digits after the final P :-(. Here **n** is the color (0-16), and *rrggbb* indicates the red/green/blue values (0-255). ESC] R: reset palette

ECMA-48 CSI sequences

CSI (or ESC [) is followed by a sequence of parameters, at most NPAR (16), that are decimal numbers separated by semicolons. An empty or absent parameter is taken to be 0. The sequence of parameters may be preceded by a single question mark.

However, after CSI [(or ESC [[) a single character is read and this entire sequence is ignored. (The idea is to ignore an echoed function key.)

The action of a CSI sequence is determined by its final character.

l l l. @ ICH Insert the indicated # of blank characters. A CUU Move cursor up the indicated # of rows. B CUD Move cursor down the indicated # of rows. C CUF Move cursor right the indicated # of columns. D CUB Move cursor left the indicated # of columns. E CNL Move cursor down the indicated # of rows, to column 1. F CPL Move cursor up the indicated # of rows, to column 1. G CHA Move cursor to indicated column in current row. H CUP Move cursor to the indicated row, column (origin at 1,1). J ED Erase display (default: from cursor to end of display). ESC [1 J: erase from start to cursor. ESC [2 J: erase whole display. K EL Erase line (default: from cursor to end of line). ESC [1 K: erase from start of line to cursor. ESC [2 K: erase whole line. L IL Insert the indicated # of blank lines. M DL Delete the indicated # of lines. P DCH Delete the indicated # of characters on the current line. X ECH Erase the indicated # of characters on the current line. a HPR Move cursor right the indicated # of columns. c DA Answer ESC [? 6 c: `I am a VT102'. d VPA Move cursor to the indicated row, current column. e VPR Move cursor down the indicated # of rows. f HVP Move cursor to the indicated row, column.

g TBC Without parameter: clear tab stop at the current position. ESC [3 g: delete all tab stops. h SM Set Mode (see below). l RM Reset Mode (see below). m SGR Set attributes (see below). n DSR Status report (see below). q DECLL Set keyboard LEDs. ESC [0 q: clear all LEDs ESC [1 q: set Scroll Lock LED ESC [2 q: set Num Lock LED ESC [3 q: set Caps Lock LED r DECSTBM Set scrolling region; parameters are top and bottom row. s ? Save cursor location. u ? Restore cursor location. ` HPA Move cursor to indicated column in current row.

ECMA-48 Set Graphics Rendition

The ECMA-48 SGR sequence ESC [<parameters> m sets display attributes. Several attributes can be set in the same sequence.

l l. par result 0 reset all attributes to their defaults 1 set bold 2 set half-bright (simulated with color on a color display) 4 set underscore (simulated with color on a color display) (the colors used to simulate dim or underline are set using ESC] ...) 5 set blink 7 set reverse video 10 reset selected mapping, display control flag, and toggle meta flag. 11 select null mapping, set display control flag, reset toggle meta flag. 12 select null mapping, set display control flag, set toggle meta flag. (The toggle meta flag causes the high bit of a byte to be toggled before the mapping table translation is done.) 21 set normal intensity (this is not compatible with ECMA-48) 22 set normal intensity 24 underline off 25 blink off 27 reverse video off 30 set black foreground 31 set red foreground 32 set green foreground 33 set brown foreground 34 set blue foreground 35 set magenta foreground 36 set cyan foreground 37 set white foreground 38 set underscore on, set default foreground color 39 set underscore off, set default foreground color 40 set black background 41 set red background 42 set green background 43 set brown background 44 set blue background 45 set magenta background 46 set cyan background 47 set white background 49 set default background color

ECMA-48 Mode Switches

ESC [3 h

DECCRM (default off): Display control chars.

ESC [4 h
DECIM (default off): Set insert mode.

ESC [20 h
LF/NL (default off): Automatically follow echo of LF, VT or FF with CR.

ECMA-48 Status Report Commands

ESC [5 n
Device status report (DSR): Answer is ESC [0 n (Terminal OK).

ESC [6 n
Cursor position report (CPR): Answer is ESC [**y** ; **x** R, where **x**,**y** is the cursor location.

DEC Private Mode (DECSET/DECRST) sequences.

These are not described in ECMA-48. We list the Set Mode sequences; the Reset Mode sequences are obtained by replacing the final `h' by `l'.

ESC [? 1 h
DECCKM (default off): When set, the cursor keys send an ESC O prefix, rather than ESC [.

ESC [? 3 h
DECCOLM (default off = 80 columns): 80/132 col mode switch. The driver sources note that this alone does not suffice; some user-mode utility such as *resizecons*(8) has to change the hardware registers on the console video card.

ESC [? 5 h
DECSCNM (default off): Set reverse-video mode.

ESC [? 6 h
DECOM (default off): When set, cursor addressing is relative to the upper left corner of the scrolling region.

ESC [? 7 h
DECAWM (default on): Set autowrap on. In this mode, a graphic character emitted after column 80 (or column

132 of DECCOLM is on) forces a wrap to the beginning of the following line first.

ESC [? 8 h
DECARM (default on): Set keyboard autorepreat on.

ESC [? 9 h
X10 Mouse Reporting (default off): Set reporting mode to 1 (or reset to 0) - see below.

ESC [? 25 h
DECCM (default on): Make cursor visible.

ESC [? 1000 h
X11 Mouse Reporting (default off): Set reporting mode to 2 (or reset to 0) - see below.

Linux Console Private CSI Sequences

The following sequences are neither ECMA-48 nor native VT102. They are native to the Linux console driver. Colors are in SGR parameters: 0 = black, 1 = red, 2 = green, 3 = brown, 4 = blue, 5 = magenta, 6 = cyan, 7 = white.

l l. ESC [1 ; **n**] Set color **n** as the underline color ESC [2 ; **n**] Set color **n** as the dim color ESC [8] Make the current color pair the default attributes. ESC [9 ; **n**] Set screen blank timeout to **n** minutes. ESC [10 ; **n**] Set bell frequency in Hz. ESC [11 ; **n**] Set bell duration in msec. ESC [12 ; **n**] Bring specified console to the front. ESC [13] Unblank the screen. ESC [14 ; **n**] Set the VESA powerdown interval in minutes.

CHARACTER SETS

The kernel knows about 4 translations of bytes into console-screen symbols. The four tables are: a) Latin1 -> PC, b) VT100 graphics -> PC, c) PC -> PC, d) user-defined.

There are two character sets, called G0 and G1, and one of them is the current character set. (Initially G0.) Typing

^N causes G1 to become current, ^O causes G0 to become current.

These variables G0 and G1 point at a translation table, and can be changed by the user. Initially they point at tables a) and b), respectively. The sequences ESC (B and ESC (0 and ESC (U and ESC (K cause G0 to point at translation table a), b), c) and d), respectively. The sequences ESC) B and ESC) 0 and ESC) U and ESC) K cause G1 to point at translation table a), b), c) and d), respectively.

The sequence ESC c causes a terminal reset, which is what you want if the screen is all garbled. The oft-advised "echo ^V^O" will only make G0 current, but there is no guarantee that G0 points at table a). In some distributions there is a program **reset**(1) that just does "echo ^[c". If your terminfo entry for the console is correct (and has an entry rs1=\Ec), then "tput reset" will also work.

The user-defined mapping table can be set using **mapscrn**(8). The result of the mapping is that if a symbol c is printed, the symbol s = map[c] is sent to the video memory. The bitmap that corresponds to s is found in the character ROM, and can be changed using **setfont(8).**

MOUSE TRACKING

The mouse tracking facility is intended to return xterm-compatible mouse status reports. Because the console driver has no way to know the device or type of the mouse, these reports are returned in the console input stream only when the virtual terminal driver receives a mouse update ioctl. These ioctls must be generated by a mouse-aware user-mode application such as the **gpm(8)** daemon.

Parameters for all mouse tracking escape sequences generated by *xterm* encode numeric parameters in a single character as *value*+040. For example, `!' is 1. The screen coordinate system is 1-based.

The X10 compatibility mode sends an escape sequence on but-

ton press encoding the location and the mouse button pressed. It is enabled by sending ESC [? 9 h and disabled with ESC [? 9 l. On button press, *xterm* sends ESC [M *bxy* (6 characters). Here **b** is button-1, and **x** and **y** are the x and y coordinates of the mouse when the button was pressed. This is the same code the kernel also produces.

Normal tracking mode (not implemented in Linux 2.0.24) sends an escape sequence on both button press and release. Modifier information is also sent. It is enabled by sending ESC [? 1000 h and disabled with ESC [1000 l. On button press or release, *xterm* sends ESC [M *bxy*. The low two bits of **b** encode button information: 0=MB1 pressed, 1=MB2 pressed, 2=MB3 pressed, 3=release. The upper bits encode what modifiers were down when the button was pressed and are added together: 4=Shift, 8=Meta, 16=Control. Again **x** and **y** are the x and y coordinates of the mouse event. The upper left corner is (1,1).

COMPARISONS WITH OTHER TERMINALS

Many different terminal types are described, like the Linux console, as being `VT100-compatible'. Here we discuss differences vbetween the Linux console an the two most important others, the DEC VT102 and *xterm*(1).

Control-character handling

The vt102 also recognized the following control characters:

NUL (0x00) was ignored;

ENQ (0x05) triggered an answerback message;

DC1 (0x11, ^Q, XON) resumed transmission;

DC3 (0x13, ^S, XOFF) caused vt100 to ignore (and stop transmitting) all codes except XOFF and XON.

VT100-like DC1/DC3 processing may be enabled by the tty driver.

The *xterm* program (in vt100 mode) recognizes the control characters BEL, BS, HT, LF, VT, FF, CR, SO, SI, ESC.

Escape sequences

VT100 console sequences not implemented on the Linux console:

l l l. ESC N SS2 Single shift 2. (Select G2 character set for the next character only.) ESC O SS3 Single shift 3. (Select G3 character set for the next character only.) ESC P DCS Device control string (ended by ESC \) ESC X SOS Start of string. ESC ^ PM Privacy message (ended by ESC \) ESC \ ST String terminator ESC * ... Designate G2 character set ESC + ... Designate G3 character set

The program *xterm* (in vt100 mode) recognizes ESC c, ESC # 8, ESC >, ESC =, ESC D, ESC E, ESC H, ESC M, ESC N, ESC O, ESC P ... ESC , ESC Z (it answers ESC [? 1 ; 2 c, `I am a vt100 with advanced video option') and ESC ^ ... ESC with the same meanings as indicated above. It accepts ESC (, ESC), ESC *, ESC + followed by 0, A, B for the DEC special character and line drawing set, UK, and USASCII, respectively. It accepts ESC] for the setting of certain resources:

l l. ESC] 0 ; txt BEL Set icon name and window title to txt. ESC] 1 ; txt BEL Set icon name to txt. ESC] 2 ; txt BEL Set window title to txt. ESC] 4 6 ; name BEL Change log file to name (normally disabled by a compile-time option) ESC] 5 0 ; fn BEL Set font to fn.

It recognizes the following with slightly modified meaning:

l l l. ESC 7 DECSC Save cursor ESC 8 DECRC Restore cursor

It also recognizes

l l l. ESC F Cursor to lower left corner of screen (if enabled by the hpLowerleftBugCompat resource) ESC l Memory lock (per HP terminals). Locks memory above the cursor. ESC m Memory unlock (per HP terminals). ESC n LS2 Invoke the G2 character set. ESC

o LS3 Invoke the G3 character set. ESC | LS3R Invoke the G3 character set as GR. Has no visible effect in xterm. ESC } LS2R Invoke the G2 character set as GR. Has no visible effect in xterm. ESC ~ LS1R Invoke the G1 character set as GR. Has no visible effect in xterm.

It does not recognize ESC % ...

CSI Sequences

The *xterm* program (as of XFree86 3.1.2G) does not recognize the blink or invisible-mode SGRs. Stock X11R6 versions do not recognize the color-setting SGRs. All other ECMA-48 CSI sequences recognized by Linux are also recognized by *xterm*, and vice-versa.

The *xterm* program will recognize all of the DEC Private Mode sequences listed above, but none of the Linux private-mode sequences. For discussion of *xterm*'s own private-mode sequences, refer to the *Xterm Control Sequences* document by Edward Moy and Stephen Gildea, available with the X distribution.

BUGS

In 2.0.23, CSI is broken, and NUL is not ignored inside escape sequences.

SEE ALSO

console(4), **console_ioctl**(4), **charsets**(4)

TOC

NAME

console ioctl - ioctl's for console terminal and virtual consoles

DESCRIPTION

WARNING: If you use the following information you are going to burn yourself.

WARNING: ioctl's are undocumented Linux internals, liable to be changed without warning. Use POSIX functions.

The following Linux-peculiar **ioctl()** requests are supported. Each requires a third argument, assumed here to be *argp*.

KDGETLED
Get state of LEDs. *argp* points to a long int. The lower three bits of **argp* are set to the state of the LEDs, as follows:

LED_CAP	0x04	caps lock led
LEC_NUM	0x02	num lock led
LED_SCR	0x01	scroll lock led

KDSETLED
Set the LEDs. The LEDs are set to correspond to the lower three bits of *argp*. However, if a higher order bit is set, the LEDs revert to normal: displaying the state of the keyboard functions of caps lock, num lock, and scroll lock.

Before 1.1.54, the LEDs just reflected the state of the corresponding keyboard flags, and KDGETLED/KDSETLED would also change the keyboard flags. Since 1.1.54 the leds can be made to display arbitrary information, but by default they display the keyboard flags. The following two ioctl's are used to access the keyboard flags.

KDGKBLED
Get keyboard flags CapsLock, NumLock, ScrollLock (not lights). *argp* points to a char which is set to the flag state. The low order three bits (mask 0x7) get the current flag state, and the low order bits of the next nibble (mask 0x70) get the default flag state.

(Since 1.1.54.)

KDSKBLED
Set keyboard flags CapsLock, NumLock, ScrollLock (not lights). *argp* has the desired flag state. The low order three bits (mask 0x7) have the flag state, and the low order bits of the next nibble (mask 0x70) have the default flag state. (Since 1.1.54.)

KDGKBTYPE
Get keyboard type. This returns the value KB_101, defined as 0x02.

KDADDIO
Add I/O port as valid. Equivalent to ioperm(arg,1,1).

KDDELIO
Delete I/O port as valid. Equivalent to ioperm(arg,1,0).

KDENABIO
Enable I/O to video board. Equivalent to ioperm(0x3b4, 0x3df-0x3b4+1, 1).

KDDISABIO
Disable I/O to video board. Equivalent to ioperm(0x3b4, 0x3df-0x3b4+1, 0).

KDSETMODE
Set text/graphics mode. *argp* is one of these:

```
KD_TEXT       0x00
KD_GRAPHICS   0x01
```

KDGETMODE
Get text/graphics mode. *argp* points to a long which is set to one of the above values.

KDMKTONE
Generate tone of specified length. The lower 16 bits of *argp* specify the period in clock cycles, and the upper 16 bits give the duration in msec. If the duration is zero, the sound is turned off. Control returns immediately. For example, *argp* = (125<<16) + 0x637 would specify the beep normally associated with a ctrl-G. (Thus since 0.99pl1; broken in 2.1.49-50.)


TOC

KIOCSOUND

Start or stop sound generation. The lower 16 bits of *argp* specify the period in clock cycles (that is, *argp* = 1193180/frequency). *argp* = 0 turns sound off. In either case, control returns immediately.

GIO_CMAP

Get the current default colour map from kernel. *argp* points to a 48-byte array. (Since 1.3.3.)

PIO_CMAP

Change the default text-mode colour map. *argp* points to a 48-byte array which contains, in order, the Red, Green, and Blue values for the 16 available screen colours: 0 is off, and 255 is full intensity. The default colours are, in order: black, dark red, dark green, brown, dark blue, dark purple, dark cyan, light grey, dark grey, bright red, bright green, yellow, bright blue, bright purple, bright cyan and white. (Since 1.3.3.)

GIO_FONT

Gets 256-character screen font in expanded form. *argp* points to an 8192 byte array. Fails with error code **EINVAL** if the currently loaded font is a 512-character font, or if the console is not in text mode.

GIO_FONTX

Gets screen font and associated information. *argp* points to a struct consolefontdesc (see **PIO_FONTX**). On call, the *charcount* field should be set to the maximum number of characters that would fit in the buffer pointed to by *chardata*. On return, the *charcount* and *charheight* are filled with the respective data for the currently loaded font, and the *chardata* array contains the font data if the initial value of *charcount* indicated enough space was available; otherwise the buffer is untouched and *errno* is set to **ENOMEM**. (Since 1.3.1.)

PIO_FONT

Sets 256-character screen font. Load font into the EGA/VGA character generator. *argp* points to a 8192 byte map, with 32 bytes per character. Only first **N** of them are used for an 8x**N** font (0 < **N** <= 32). This call also invalidates the Unicode mapping.

PIO_FONTX

Sets screen font and associated rendering information. *argp* points to a

```
struct consolefontdesc {
        u_short charcount;      /* characters in font (256 or 512) */
        u_short charheight;     /* scan lines per character (1-32) */
        char *chardata;         /* font data in expanded form */
};
```

If necessary, the screen will be appropriately resized, and **SIGWINCH** sent to the appropriate processes. This call also invalidates the Unicode mapping. (Since 1.3.1.)

PIO_FONTRESET

Resets the screen font, size and Unicode mapping to the bootup defaults. *argp* is unused, but should be set to **NULL** to ensure compatibility with future versions of Linux. (Since 1.3.28.)

GIO_SCRNMAP

Get screen mapping from kernel. *argp* points to an area of size E_TABSZ, which is loaded with the font positions used to display each character. This call is likely to return useless information if the currently loaded font is more than 256 characters.

GIO_UNISCRNMAP

Get full Unicode screen mapping from kernel. *argp* points to an area of size E_TABSZ*sizeof(unsigned short), which is loaded with the Unicodes each character represent. A special set of Unicodes, starting at U+F000, are used to represent ``direct to font'' mappings. (Since 1.3.1.)

PIO_SCRNMAP

Loads the ``user definable'' (fourth) table in the kernel which maps bytes into console screen symbols. *argp* points to an area of size E_TABSZ.

PIO_UNISCRNMAP

Loads the ``user definable'' (fourth) table in the kernel which maps bytes into Unicodes, which are then translated into screen symbols according to the currently loaded Unicode-to-font map. Special Unicodes starting at U+F000 can be used to map directly to the font symbols. (Since 1.3.1.)

GIO_UNIMAP

Get Unicode-to-font mapping from kernel. *argp* points to a

```
struct unimapdesc {
        u_short entry_ct;
        struct unipair *entries;
};
```

where *entries* points to an array of

```
struct unipair {
        u_short unicode;
        u_short fontpos;
};
```

(Since 1.1.92.)

PIO_UNIMAP

Put unicode-to-font mapping in kernel. *argp* points to a struct unimapdesc. (Since 1.1.92)

PIO_UNIMAPCLR

Clear table, possibly advise hash algorithm. *argp* points to a

```
struct unimapinit {
        u_short advised_hashsize;  /* 0 if no opinion */
        u_short advised_hashstep;  /* 0 if no opinion */
        u_short advised_hashlevel; /* 0 if no opinion */
};
```

(Since 1.1.92.)

KDGKBMODE

Gets current keyboard mode. *argp* points to a long which is set to one of these:

```
K_RAW         0x00
K_XLATE       0x01
K_MEDIUMRAW   0x02
K_UNICODE     0x03
```

KDSKBMODE

Sets current keyboard mode. *argp* is a long equal to one of the above values.

KDGKBMETA

Gets meta key handling mode. *argp* points to a long which is set to one of these:


```
    K_METABIT     0x03   set high order bit
    K_ESCPREFIX   0x04   escape prefix
```

KDSKBMETA

Sets meta key handling mode. *argp* is a long equal to one of the above values.

KDGKBENT

Gets one entry in key translation table (keycode to action code). *argp* points to a

```
struct kbentry {
    u_char kb_table;
    u_char kb_index;
    u_short kb_value;
};
```

with the first two members filled in: *kb_table* selects the key table (0 <= *kb_table* < MAX_NR_KEYMAPS), and *kb_index* is the keycode (0 <= *kb_index* < NR_KEYS). *kb_value* is set to the corresponding action code, or K_HOLE if there is no such key, or K_NOSUCHMAP if *kb_table* is invalid.

KDSKBENT

Sets one entry in translation table. *argp* points to a struct kbentry.

KDGKBSENT

Gets one function key string. *argp* points to a

```
struct kbsentry {
    u_char kb_func;
    u_char kb_string[512];
};
```

kb_string is set to the (NULL terminated) string corresponding to the *kb_func*th function key action code.

KDSKBSENT

Sets one function key string entry. *argp* points to a struct kbsentry.

KDGKBDIACR

Read kernel accent table. *argp* points to a

```
struct kbdiacrs {
    unsigned int kb_cnt;
    struct kbdiacr kbdiacr[256];
```


```
};
```

where *kb_cnt* is the number of entries in the array, each of which is a

```
struct kbdiacr { u_char diacr, base, result; };
```

KDGETKEYCODE

Read kernel keycode table entry (scan code to keycode). *argp* points to a

```
struct kbkeycode { unsigned int scancode, keycode; };
```

keycode is set to correspond to the given *scancode*. (89 <= *scancode* <= 255 only. For 1 <= *scancode* <= 88, *keycode*==*scancode*.) (Since 1.1.63.)

KDSETKEYCODE

Write kernel keycode table entry. *argp* points to struct kbkeycode. (Since 1.1.63.)

KDSIGACCEPT

The calling process indicates its willingness to accept the signal *argp* when it is generated by pressing an appropriate key combination. (1 <= *argp* <= NSIG). (See spawn_console() in linux/drivers/char/keyboard.c.)

VT_OPENQRY

Returns the first available (non-opened) console. *argp* points to an int which is set to the number of the vt (1 <= **argp* <= MAX_NR_CONSOLES).

VT_GETMODE

Get mode of active vt. *argp* points to a

```
struct vt_mode {
    char mode;     /* vt mode */
    char waitv;    /* if set, hang on writes if not active */
    short relsig;  /* signal to raise on release req */
    short acqsig;  /* signal to raise on acquisition */
    short frsig;   /* unused (set to 0) */

};
```

mode is set to one of these values:

```
    VT_AUTO       auto vt switching
    VT_PROCESS    process controls switching
    VT_ACKACQ     acknowledge switch
```

VT_SETMODE
Set mode of active vt. *argp* points to a struct vt_mode.

VT_GETSTATE
Get global vt state info. *argp* points to a

```
struct vt_stat {
    ushort v_active;  /* active vt */
    ushort v_signal;  /* signal to send */
    ushort v_state;   /* vt bitmask */
};
```

For each vt in use, the corresponding bit in the *v_state* member is set. (Kernels 1.0 through 1.1.92.)

VT_RELDISP
Release a display.

VT_ACTIVATE
Switch to vt *argp* (1 <= *argp* <= MAX_NR_CONSOLES).

VT_WAITACTIVE
Wait until vt *argp* has been activated.

VT_DISALLOCATE
Deallocate the memory associated with vt *argp*. (Since 1.1.54.)

VT_RESIZE
Set the kernel's idea of screensize. *argp* points to a

```
struct vt_sizes {
    ushort v_rows;       /* # rows */
    ushort v_cols;       /* # columns */
    ushort v_scrollsize; /* no longer used */
};
```

Note that this does not change the videomode. See resizecons(8). (Since 1.1.54.)

VT_RESIZEX
Set the kernel's idea of various screen parameters. *argp* points to a

```
struct vt_consize {
        ushort v_rows;          /* number of rows */
        ushort v_cols;          /* number of columns */
```


```
        ushort v_vlin;          /* number of pixel rows on screen */
        ushort v_clin;          /* number of pixel rows per character */
        ushort v_vcol;          /* number of pixel columns on screen */
        ushort v_ccol;          /* number of pixel columns per character */
};
```

Any parameter may be set to zero, indicating ``no change'', but if multiple parameters are set, they must be self-consistent. Note that this does not change the videomode. See resizecons(8). (Since 1.3.3.)

The action of the following ioctls depends on the first byte in the struct pointed to by *argp*, referred to here as the *subcode*. These are legal only for the superuser or the owner of the current tty.

TIOCLINUX, subcode=0
Dump the screen. Disappeared in 1.1.92. (With kernel 1.1.92 or later, read from /dev/vcsN or /dev/vcsaN instead.)

TIOCLINUX, subcode=1
Get task information. Disappeared in 1.1.92.

TIOCLINUX, subcode=2
Set selection. *argp* points to a

```
struct {char subcode;
    short xs, ys, xe, ye;
    short sel_mode;
}
```

xs and *ys* are the starting column and row. *xe* and *ye* are the ending column and row. (Upper left corner is row=column=1.) *sel_mode* is 0 for character-by-character selection, 1 for word-by-word selection, or 2 for line-by-line selection. The indicated screen characters are highlighted and saved in the static array sel_buffer in devices/char/console.c.

TIOCLINUX, subcode=3
Paste selection. The characters in the selection buffer are written to *fd*.

TIOCLINUX, subcode=4
Unblank the screen.

TIOCLINUX, subcode=5
Sets contents of a 256-bit look up table defining char-

acters in a "word", for word-by-word selection. (Since 1.1.32.)

TIOCLINUX, subcode=6
argp points to a char which is set to the value of the kernel variable *shift_state*. (Since 1.1.32.)

TIOCLINUX, subcode=7
argp points to a char which is set to the value of the kernel variable *report_mouse*. (Since 1.1.33.)

TIOCLINUX, subcode=8
Dump screen width and height, cursor position, and all the character-attribute pairs. (Kernels 1.1.67 through 1.1.91 only. With kernel 1.1.92 or later, read from /dev/vcsa* instead.)

TIOCLINUX, subcode=9
Restore screen width and height, cursor position, and all the character-attribute pairs. (Kernels 1.1.67 through 1.1.91 only. With kernel 1.1.92 or later, write to /dev/vcsa* instead.)

TIOCLINUX, subcode=10
Handles the Power Saving feature of the new generation of monitors. VESA screen blanking mode is set to *argp*[1], which governs what screen blanking does:

0: Screen blanking is disabled.

1: The current video adapter register settings are saved, then the controller is programmed to turn off the vertical synchronization pulses. This puts the monitor into "standby" mode. If your monitor has an Off_Mode timer, then it will eventually power down by itself.

2: The current settings are saved, then both the vertical and horizontal synchronization pulses are turned off. This puts the monitor into "off" mode. If your monitor has no Off_Mode timer, or if you want your monitor to power down immediately when the blank_timer times out, then you choose this option. (*Caution*: Powering down frequently will damage the monitor.)

(Since 1.1.76.)

RETURN VALUES

-1 for error, and *errno* is set.

ERRORS

errno may take on these values:

EBADF
file descriptor is invalid.

ENOTTY
file descriptor is not associated with a character special device, or the specified request does not apply to it.

EINVAL
file descriptor or *argp* is invalid.

EPERM
permission violation.

WARNING

Do not regard this man page as documentation of the Linux console ioctl's. This is provided for the curious only, as an alternative to reading the source. Ioctl's are undocumented Linux internals, liable to be changed without warning. (And indeed, this page more or less describes the situation as of kernel version 1.1.94; there are many minor and not-so-minor differences with earlier versions.)

Very often, ioctl's are introduced for communication between the kernel and one particular well-known program (fdisk, hdparm, setserial, tunelp, loadkeys, selection, setfont, etc.), and their behavior will be changed when required by this particular program.

Programs using these ioctl's will not be portable to other versions of Unix, will not work on older versions of Linux, and will not work on future versions of Linux.

Use POSIX functions.

TOC

SEE ALSO

kbd_mode(1), **loadkeys**(1), **dumpkeys**(1), **mknod**(1), **setleds**(1), **setmetamode**(1), **ioperm**(2), **termios**(2), **execve**(2), **fcntl**(2), **charsets**(4), **console**(4), **console_codes**(4), **mt**(4), **sd**(4), **tty**(4), **ttys**(4), **vcs**(4), **vcsa**(4), **mapscrn**(8), **setfont**(8), **resizecons**(8), */usr/include/linux/kd.h*, */usr/include/linux/vt.h*

TOC

NAME

fd - floppy disk device

CONFIGURATION

Floppy drives are block devices with major number 2. Typically they are owned by root.floppy (i.e., user root, group floppy) and have either mode 0660 (access checking via group membership) or mode 0666 (everybody has access). The minor numbers encode the device type, drive number, and controller number. For each device type (that is, combination of density and track count) there is a base minor number. To this base number, add the drive's number on its controller and 128 if the drive is on the secondary controller. In the following device tables, **n** represents the drive number

Warning: If you use formats with more tracks than supported by your drive, you may cause it mechanical damage. Trying once if more tracks than the usual 40/80 are supported should not damage it, but no warranty is given for that. Don't create device entries for those formats to prevent their usage if you are not sure.

Drive independent device files which automatically detect the media format and capacity:

l l. Name Base minor # _ **fdn** 0

5.25 inch double density device files:

lw(1i) l l l l l.
Name Capac. Cyl. Sect. Heads Base minor # _
fdnd360 360K 40 9 2 4

5.25 inch high density device files:

lw(1i) Name	Capac.	l Cyl.	l Sect.	l Heads	l Base	l minor	l #	l. _
fdnh360	360K	40	9	2	20			
fdnh410	410K	41	10	2	48			
fdnh420	420K	42	10	2	64			
fdnh720	720K	80	9	2	24			
fdnh880	880K	80	11	2	80			
fdnh1200	1200K		80	15	2	8		
fdnh1440	1440K		80	18	2	40		
fdnh1476	1476K		82	18	2	56		
fdnh1494	1494K		83	18	2	72		
fdnh1600	1600K		80	20	2	92		

3.5 inch double density device files:

lw(1i) Name	Capac.	l Cyl.	l Sect.	l Heads	l Base	l minor	l #	l. _
fdnD360	360K	80	9	1	12			
fdnD720	720K	80	9	2	16			
fdnD800	800K	80	10	2	120			
fdnD1040	1040K		80	13	2	84		
fdnD1120	1120K		80	14	2	88		

3.5 inch high density device files:

lw(1i) Name	Capac.	l Cyl.	l Sect.	l Heads	l Base	l minor	l #	l. _
fdnH360	360K	40	9	2	12			
fdnH720	720K	80	9	2	16			
fdnH820	820K	82	10	2	52			
fdnH830	830K	83	10	2	68			
fdnH1440	1440K		80	18	2	28		
fdnH1600	1600K		80	20	2	124		
fdnH1680	1680K		80	21	2	44		
fdnH1722	1722K		82	21	2	60		
fdnH1743	1743K		83	21	2	76		
fdnH1760	1760K		80	22	2	96		
fdnH1840	1840K		80	23	2	116		
fdnH1920	1920K		80	24	2	100		

3.5 inch extra density device files:

lw(1i) Name	Capac.	l Cyl.	l Sect.	l Heads	l Base	l minor	l #	l. _
fdnE2880	2880K		80	36	2	32		
fdnCompaQ	2880K		80	36	2	36		

fdnE3200	3200K	80	40	2	104
fdnE3520	3520K	80	44	2	108
fdnE3840	3840K	80	48	2	112

DESCRIPTION

fd special files access the floppy disk drives in raw mode. The following *ioctl*(2) calls are supported by **fd** devices:

FDCLRPRM
clears the media information of a drive (geometry of disk in drive).

FDSETPRM
sets the media information of a drive. The media information will be lost when the media is changed.

FDDEFPRM
sets the media information of a drive (geometry of disk in drive). The media information will not be lost when the media is changed. This will disable autodetection. In order to re-enable autodetection, you have to issue an **FDCLRPRM** .

FDGETDRVTYP
returns the type of a drive (name parameter). For formats which work in several drive types, **FDGETDRVTYP** returns a name which is appropriate for the oldest drive type which supports this format.

FDFLUSH
invalidates the buffer cache for the given drive.

FDSETMAXERRS
sets the error thresholds for reporting errors, aborting the operation, recalibrating, resetting, and reading sector by sector.

FDSETMAXERRS
gets the current error thresholds.

FDGETDRVTYP

gets the internal name of the drive.

FDWERRORCLR
clears the write error statistics.

FDWERRORGET
reads the write error statistics. These include the total number of write errors, the location and disk of the first write error, and the location and disk of the last write error. Disks are identified by a generation number which is incremented at (almost) each disk change.

FDTWADDLE
Switch the drive motor off for a few microseconds. This might be needed in order to access a disk whose sectors are too close together.

FDSETDRVPRM
sets various drive parameters.

FDGETDRVPRM
reads these parameters back.

FDGETDRVSTAT
gets the cached drive state (disk changed, write protected et al.)

FDPOLLDRVSTAT
polls the drive and return its state.

FDGETFDCSTAT
gets the floppy controller state.

FDRESET
resets the floppy controller under certain conditions.

FDRAWCMD
sends a raw command to the floppy controller.

For more precise information, consult also the <linux/fd.h> and <linux/fdreg.h> include files, as well as the manual page for floppycontrol.

NOTES

The various formats allow to read and write many types of disks. However, if a floppy is formatted with a too small inter sector gap, performance may drop, up to needing a few seconds to access an entire track. To prevent this, use interleaved formats. It is not possible to read floppies which are formatted using GCR (group code recording), which is used by Apple II and Macintosh computers (800k disks). Reading floppies which are hard sectored (one hole per sector, with the index hole being a little skewed) is not supported. This used to be common with older 8 inch floppies.

FILES

/dev/fd*

AUTHORS

Alain Knaff (Alain.Knaff@imag.fr), David Niemi (niemidc@clark.net), Bill Broadhurst (bbroad@netcom.com).

SEE ALSO

floppycontrol(1), **mknod**(1), **chown**(1), **getfdprm**(1), **superformat**(1), **mount**(8), **setfdprm**(8)


TOC

NAME

full - always full device

DESCRIPTION

File **/dev/full** has major device number 1 and minor device number 7.

Writes to the **/dev/full** device will fail with an ENOSPC error.

Reads from the **/dev/full** device will return \0 characters.

Seeks on **/dev/full** will always succeed.

CONFIGURING

If your system does not have **/dev/full** created already, it can be created with the following commands:

```
mknod -m 666 /dev/full c 1 7
chown root.root /dev/full
```

FILES

/dev/full

SEE ALSO

mknod(1), **null**(4), **zero**(4)

NAME

hd - MFM/IDE hard disk devices

DESCRIPTION

Hd* are block devices to access MFM/IDE hard disk drives in raw mode. The master drive on the primary IDE controller (major device number 3) is **hda**; the slave drive is **hdb**. The master drive of the second controller (major device number 22) is **hdc** and the slave **hdd**.

General IDE block device names have the form **hdX**, or **hd***XP*, where **X** is a letter denoting the physical drive, and **P** is a number denoting the partition on that physical drive. The first form, **hdX**, is used to address the whole drive. Partition numbers are assigned in the order the partitions are discovered, and only non-empty, non-extended partitions get a number. However, partition numbers 1-4 are given to the four partitions described in the MBR (the `primary' partitions), regardless of whether they are unused or extended. Thus, the first logical partition will be **hdX5**. Both DOS-type partitioning and BSD-disklabel partitioning are supported. You can have at most 63 partitions on an IDE disk.

For example, **/dev/hda** refers to all of the first IDE drive in the system; and **/dev/hdb3** refers to the third DOS `primary' partition on the second one.

They are typically created by:

```
mknod -m 660 /dev/hda b 3 0
mknod -m 660 /dev/hda1 b 3 1
mknod -m 660 /dev/hda2 b 3 2
...
mknod -m 660 /dev/hda8 b 3 8
mknod -m 660 /dev/hdb b 3 64
```


```
mknod -m 660 /dev/hdb1 b 3 65
mknod -m 660 /dev/hdb2 b 3 66
...
mknod -m 660 /dev/hdb8 b 3 72
chown root.disk /dev/hd*
```

FILES

/dev/hd*

SEE ALSO

mknod(1), **chown**(1), **mount**(8),

TOC

NAME

initrd - boot loader initialized RAM disk

DESCRIPTION

The special file **/dev/initrd** is a read-only block device. Device **/dev/initrd** is a RAM disk that is initialized (e.g. loaded) by the boot loader before the kernel is started. The kernel then can use the the block device **/dev/initrd**'s contents for a two phased system boot-up.

In the first boot-up phase, the kernel starts up and mounts an initial root file-system from the contents of **/dev/initrd** (e.g. RAM disk initialized by the boot loader). In the second phase, additional drivers or other modules are loaded from the initial root device's contents. After loading the additional modules, a new root file system (i.e. the normal root file system) is mounted from a different device.

BOOT-UP OPERATION

When booting up with **initrd**, the system boots as follows:

1. The boot loader loads the kernel program and **/dev/initrd**'s contents into memory.

2. On kernel startup, the kernel uncompresses and copies the contents of the device **/dev/initrd** onto device **/dev/ram0** and then frees the memory used by **/dev/initrd**.

3. The kernel then read-write mounts device **/dev/ram0** as the initial root file system.

4. If the indicated normal root file system is also the initial root file-system (e.g. **/dev/ram0**) then the kernel skips to the last step for the usual boot sequence.

5. If the executable file **/linuxrc** is present in the initial root file-system, **/linuxrc** is executed with uid 0. (The file **/linuxrc** must have executable permission. The file **/linuxrc** can be any valid executable, including a shell script.)

6. If **/linuxrc** is not executed or when **/linuxrc** terminates, the normal root file system is mounted. (If **/linuxrc** exits with any file-systems mounted on the initial root file-system, then the behavior of the kernel is **UNSPECIFIED**. See the **NOTES** section for the current kernel behavior.)

7. If the normal root file has directory **/initrd**, device **/dev/ram0** is moved from **/** to **/initrd**. Otherwise if directory **/initrd** does not exist device **/dev/ram0** is unmounted. (When moved from **/** to **/initrd**, **/dev/ram0** is not unmounted and therefore processes can remain running from **/dev/ram0**. If directory **/initrd** does not exist on the normal root file-system and any processes remain running from **/dev/ram0** when **/linuxrc** exits, the behavior of the kernel is **UNSPECIFIED**. See the **NOTES** section for the current kernel behavior.)

8. The usual boot sequence (e.g. invocation of **/sbin/init**) is performed on the normal root file system.

OPTIONS

The following boot loader options when used with **initrd**, affect the kernel's boot-up operation:

initrd=*filename*

Specifies the file to load as the contents of **/dev/initrd**. For **LOADLIN** this is a command line option. For **LILO** you have to use this command in the **LILO** configuration file **/etc/lilo.config**. The filename specified with this option will typically be a gzipped

file-system image.

noinitrd
This boot time option disables the two phase boot-up operation. The kernel performs the usual boot sequence as if **/dev/initrd** was not initialized. With this option, any contents of **/dev/initrd** loaded into memory by the boot loader contents are preserved. This option permits the contents of **/dev/initrd** to be any data and need not be limited to a file system image. However, device **/dev/initrd** is read-only and can be read only one time after system startup.

root=*device-name*
Specifies the device to be used as the normal root file system. For **LOADLIN** this is a command line option. For **LILO** this is a boot time option or can be used as an option line in the **LILO** configuration file **/etc/lilo.config**. The device specified by the this option must be a mountable device having a suitable root file-system.

CHANGING THE NORMAL ROOT FILE SYSTEM

By default, the kernel's settings (e.g. set in the kernel file with **rdev** or compiled into the kernel file), or the boot loader option setting is used for the normal root file systems. For a NFS-mounted normal root file system, one has to use the **nfs_root_name** and **nfs_root_addrs** boot options to give the NFS settings. For more information on NSF-mounted root see the kernel documentation file **nfsroot.txt**. For more information on setting the root file system also see the **LILO** and **LOADLIN** documentation.
It is also possible for the **/linuxrc** executable to change the normal root device. For **/linuxrc** to change the normal root device, **/proc** must be mounted. After mounting **/proc**, **/linuxrc** changes the normal root device by writing into the proc files **/proc/sys/kernel/real-root-dev**, **/proc/sys/kernel/nfs-root-name**, and **/proc/sys/kernel/nfs-root-addrs**. For a physical root device, the root device is changed by having **/linuxrc** write the new root file system device number into **/proc/sys/kernel/real-root-dev**. For a

NSF root file system, the root device is changed by having **/linuxrc** write the NSF setting into files **/proc/sys/kernel/nfs-root-name** and **/proc/sys/kernel/nfs-root-addrs** and then writing 0xff (e.g. the pseudo-NFS-device number) into file **/proc/sys/kernel/real-root-dev**. For example, the following shell command line would change the normal root device to **/dev/hdb1**:

```
echo 0x365 >/proc/sys/kernel/real-root-dev
```

For a NSF example, the following shell command lines would change the normal root device to the NSF directory **/var/nfsroot** on a local networked NSF server with IP number 193.8.232.7 for a system with IP number 193.8.232.7 and named 'idefix':

```
echo /var/nfsroot >/proc/sys/kernel/nfs-root-name
echo 193.8.232.2:193.8.232.7::255.255.255.0:idefix \
  >/proc/sys/kernel/nfs-root-addrs
echo 255 >/proc/sys/kernel/real-root-dev
```

USAGE

The main motivation for implementing **initrd** was to allow for modular kernel configuration at system installation.

A possible system installation scenario is as follows:

1. The loader program boots from floppy or other media with a minimal kernel (e.g. support for **/dev/ram**, **/dev/initrd**, and the ext2 file-system) and loads **/dev/initrd** with a gzipped version of the initial file-system.

2. The executable **/linuxrc** determines what is needed to (1) mount the normal root file-system (i.e. device type, device drivers, file system) and (2) the distribution media (e.g. CD-ROM, network, tape, ...). This can be done by asking the user, by auto-probing, or by using a hybrid approach.

3. The executable **/linuxrc** loads the necessary modules from the initial root file-system.

4. The executable **/linuxrc** creates and populates the root

file system. (At this stage the normal root file system does not have to be a completed system yet.)

5. The executable **/linuxrc** sets **/proc/sys/kernel/real-root-dev,** unmount **/proc**, the normal root file system and any other file systems it has mounted, and then terminates.

6. The kernel then mounts the normal root file system.

7. Now that the file system is accessible and intact, the boot loader can be installed.

8. The boot loader is configured to load into **/dev/initrd** a file system with the set of modules that was used to bring up the system. (e.g. Device **/dev/ram0** can be modified, then unmounted, and finally, the image is written from **/dev/ram0** to a file.)

9. The system is now bootable and additional installation tasks can be performed.

The key role of **/dev/initrd** in the above is to re-use the configuration data during normal system operation without requiring initial kernel selection, a large generic kernel or, recompiling the kernel.

A second scenario is for installations where Linux runs on systems with different hardware configurations in a single administrative network. In such cases, it may be desirable to use only a small set of kernels (ideally only one) and to keep the system-specific part of configuration information as small as possible. In this case, create a common file with all needed modules. Then, only the the **/linuxrc** file or a file executed by **/linuxrc** would be different.

A third scenario is more convenient recovery disks. Because information like the location of the root file-system partition is not needed at boot time, the system loaded from **/dev/initrd** can use a dialog and/or auto-detection followed by a possible sanity check.

Last but not least, Linux distributions on CD-ROM may use **initrd** for easy installation from the CD-ROM. The distribution can use **LOADLIN** to directly load **/dev/initrd** from CD-ROM without the need of any floppies. The distribution could also use a **LILO** boot floppy and then bootstrap a

bigger ram disk via **/dev/initrd** from the CD-ROM.

CONFIGURATION

The **/dev/initrd** is a read-only block device assigned major number 1 and minor number 250. Typically **/dev/initrd** is owned by **root.disk** with mode 0400 (read access by root only). If the Linux system does not have **/dev/initrd** already created, it can be created with the following commands:

```
mknod -m 400 /dev/initrd b 1 250
chown root.disk /dev/initrd
```

Also, support for both "RAM disk" and "Initial RAM disk" (e.g. **CONFIG_BLK_DEV_RAM=y** and **CONFIG_BLK_DEV_INITRD=y**) support must be compiled directly into the Linux kernel to use **/dev/initrd**. When using **/dev/initrd**, the RAM disk driver cannot be loaded as a module.

FILES

/dev/initrd
/dev/ram0
/linuxrc
/initrd

SEE ALSO

chown(1), **mknod**(1), **/dev/ram**(4), **freeramdisk**(8), **rdev**(8), The documentation file *initrd.txt* in the kernel source package, the LILO documentation, the LOADLIN documentation, the SYSLINUX documentation.


TOC

NOTES

1. With the current kernel, any file systems that remain mounted when **/dev/ram0** is moved from **/** to **/initrd** continue to be accessible. However, the **/proc/mounts** entries are not updated.

2. With the current kernel, if directory **/initrd** does not exist, then **/dev/ram0** will NOT be fully unmounted if **/dev/ram0** is used by any process or has any file-system mounted on it. If **/dev/ram0** is NOT fully unmounted, then **/dev/ram0** will remain in memory.

3. Users of **/dev/initrd** should not depend on the behavior give in the above notes. The behavior may change in future versions of the Linux kernel.

AUTHOR

The kernel code for device **initrd** was written by Werner Almesberger <almesber@lrc.epfl.ch> and Hans Lermen <lermen@elserv.ffm.fgan.de>. The code for **initrd** was added to the baseline Linux kernel in development version 1.3.73.

NAME

intro - Introduction to special files

DESCRIPTION

This chapter describes special files.

FILES

/dev/* - device files

AUTHORS

Look at the header of the manual page for the author(s) and copyright conditions. Note that these can be different from page to page!

TOC

NAME

lp - line printer devices

SYNOPSIS

```
#include <linux/lp.h>
```

CONFIGURATION

lp[02] are character devices for the parallel line printers; they have major number 6 and minor number 02. The minor numbers correspond to the printer port base addresses 0x03bc, 0x0378 and 0x0278. Usually they have mode 220 and are owned by root and group lp. You can use printer ports either with polling or with interrupts. Interrupts are recommended when high traffic is expected, e.g. for laser printers. For usual dot matrix printers polling will usually be enough. The default is polling.

DESCRIPTION

The following *ioctl*(2) calls are supported:

int ioctl(int *fd*, **LPTIME, int** *arg)*)

Sets the amount of time that the driver sleeps before rechecking the printer when the printer's buffer appears to be filled to *arg*. If you have a fast printer, decrease this number; if you have a slow printer then increase it. This is in hundredths of a

second, the default 2 being 0.02 seconds. It only influences the polling driver.

int ioctl(int *fd*, **LPCHAR, int** *arg)*)
Sets the maximum number of busy-wait iterations which the polling driver does while waiting for the printer to get ready for receiving a character to *arg*. If printing is too slow, increase this number; if the system gets too slow, decrease this number. The default is 1000. It only influences the polling driver.

int ioctl(int *fd*, **LPABORT, int** *arg)*)
If *arg* is 0, the printer driver will retry on errors, otherwise it will abort. The default is 0.

int ioctl(int *fd*, **LPABORTOPEN, int** *arg)*)
If *arg* is 0, *open*(2) will be aborted on error, otherwise error will be ignored. The default is to ignore it.

int ioctl(int *fd*, **LPCAREFUL, int** *arg)*)
If *arg* is 0, then the out-of-paper, offline and error signals are required to be false on all writes, otherwise they are ignored. The default is to ignore them.

int ioctl(int *fd*, **LPWAIT, int** *arg)*)
Sets the number of busy waiting iterations to wait before strobing the printer to accept a just-written character, and the number of iterations to wait before turning the strobe off again, to *arg*. The specification says this time should be 0.5 microseconds, but experience has shown the delay caused by the code is already enough. For that reason, the default value is 0. This is used for both the polling and the interrupt driver.

int ioctl(int *fd*, **LPSETIRQ, int** *arg)*)
This ioctl() requires superuser privileges. It takes an int containing the new IRQ as argument. As a side effect, the printer will be reset. When *arg* is 0, the polling driver will be used, which is also default.

int ioctl(int *fd*, **LPGETIRQ, int** ***arg)**)
Stores the currently used IRQ in *arg*.

int ioctl(int *fd*, **LPGETSTATUS, int** ***arg)**)


TOC

Stores the value of the status port in *arg*. The bits have the following meaning:

l l. LP_PBUSY inverted busy input, active high
LP_PACK unchanged acknowledge input, active low
LP_POUTPA unchanged out-of-paper input, active high
LP_PSELECD unchanged selected input, active high
LP_PERRORP unchanged error input, active low

Refer to your printer manual for the meaning of the signals. Note that undocumented bits may also be set, depending on your printer.

int ioctl(int *fd*, **LPRESET)**
Resets the printer. No argument is used.

FILES

/dev/lp*

AUTHORS

The printer driver was originally written by Jim Weigand and Linus Torvalds. It was further improved by Michael K. Johnson. The interrupt code was written by Nigel Gamble. Alan Cox modularised it. LPCAREFUL, LPABORT, LPGETSTATUS were added by Chris Metcalf.

SEE ALSO

mknod(1), **chown**(1), **chmod**(1),

NAME

mem, kmem, port - system memory, kernel memory and system ports

DESCRIPTION

Mem is a character device file that is an image of the main memory of the computer. It may be used, for example, to examine (and even patch) the system.

Byte addresses in mem are interpreted as physical memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching is likely to lead to unexpected results when read-only or write-only bits are present.

It is typically created by:

```
mknod -m 660 /dev/mem c 1 1
chown root.mem /dev/mem
```

The file kmem is the same as mem, except that the kernel virtual memory rather than physical memory is accessed.

It is typically created by:

```
mknod -m 640 /dev/kmem c 1 2
chown root.mem /dev/kmem
```

Port is similar to mem, but the IO ports are accessed.

It is typically created by:

```
mknod -m 660 /dev/port c 1 4
chown root.mem /dev/port
```

FILES

```
/dev/mem
/dev/kmem
/dev/port
```

SEE ALSO

mknod(1), **chown**(1), **ioperm**(2)

NAME

mouse - serial mouse interface

CONFIG

Serial mice are connected to a serial RS232/V24 dialout line, see *cua*(4) for a description.

DESCRIPTION

Introduction

The pinout of the usual 9 pin plug as used for serial mice is:

center; r c l. pin name used for 2 RX Data 3 TX -12 V, Imax = 10 mA 4 DTR +12 V, Imax = 10 mA 7 RTS +12 V, Imax = 10 mA 5 GND Ground

This is the specification, in fact 9 V suffices with most mice.

The mouse driver can recognize a mouse by dropping RTS to low and raising it again. About 14 ms later the mouse will send 0x4D ('M') on the data line. After a further 63 ms, a Microsoft-compatible 3-button mouse will send 0x33 ('3').

The relative mouse movement is sent as *dx* (positive means right) and *dy* (positive means down). Various mice can operate at different speeds. To select speeds, cycle through the speeds 9600, 4800, 2400 and 1200 bit/s, each time writing the two characters from the table below and waiting 0.1 seconds. The following table shows available speeds and the strings that select them:


```
center; l l.  bit/s    string  9600 *q  4800 *p  2400 *o
1200 *n
```

The first byte of a data packet can be used to synchronisation purposes.

Microsoft protocol

The **Microsoft** protocol uses 1 start bit, 7 data bits, no parity and one stop bit at the speed of 1200 bits/sec. Data is sent to RxD in 3-byte packets. The *dx* and *dy* movements are sent as two's-complement, *lb* (*rb*) are set when the left (right) button is pressed:

```
center;     r     c     c     c     c     c     c     c.
byte d6   d5   d4   d3   d2   d1   d0
1    1    lb   rb   dy7  dy6  dx7  dx6
2    0    dx5  dx4  dx3  dx2  dx1  dx0
3    0    dy5  dy4  dy3  dy2  dy1  dy0
```

3-button Microsoft protocol

Original Microsoft mice only have two buttons. However, there are some three button mice which also use the Microsoft protocol. Pressing or releasing the middle button is reported by sending a packet with zero movement and no buttons pressed. (Thus, unlike for the other two buttons, the status of the middle button is not reported in each packet.)

Logitech protocol

Logitech serial 3-button mice use a different extension of the Microsoft protocol: when the middle button is up, the above 3-byte packet is sent. When the middle button is down a 4-byte packet is sent, where the 4th byte has value 0x20 (or at least has the 0x20 bit set). In particular, a press of the middle button is reported as 0,0,0,0x20 when no other buttons are down.

Mousesystems protocol

The **Mousesystems** protocol uses 1 start bit, 8 data bits, no parity and two stop bits at the speed of 1200 bits/sec. Data is sent to RxD in 5-byte packets. *dx* is sent as the sum of the two two's-complement values, *dy* is send as negated sum of the two two's-complement values. *lb* (*mb*, *rb*) are cleared when the left (middle, right) button is pressed:

center;	r	c	c	c	c	c	c	c	c.
byte	d7	d6	d5	d4	d3	d2	d1	d0	
1	1	0	0	0	0	lb	mb	rb	
2	0	dxa6	dxa5	dxa4	dxa3	dxa2	dxa1	dxa0	
3	0	dya6	dya5	dya4	dya3	dya2	dya1	dya0	
4	0	dxb6	dxb5	dxb4	dxb3	dxb2	dxb1	dxb0	
5	0	dyb6	dyb5	dyb4	dyb3	dyb2	dyb1	dyb0	

Bytes 4 and 5 describe the change that occurred since bytes 2 and 3 were transmitted.

Sun protocol

The **Sun** protocol is the 3-byte version of the above 5-byte Mousesystems protocol: the last two bytes are not sent.

MM protocol

The **MM** protocol uses 1 start bit, 8 data bits, odd parity and one stop bit at the speed of 1200 bits/sec. Data is sent to RxD in 3-byte packets. *dx* and *dy* are sent as single signed values, the sign bit indicating a negative value. *lb* (*mb*, *rb)* are set when the left (middle, right) button is pressed:

center;	r	c	c	c	c	c	c	c	c.
byte	d7	d6	d5	d4	d3	d2	d1	d0	
1	1	0	0	dxs	dys	lb	mb	rb	
2	0	dx6	dx5	dx4	dx3	dx2	dx1	dx0	
3	0	dy6	dy5	dy4	dy3	dy2	dy1	dy0	

FILES

/dev/mouse	a commonly used symlink pointing to a mouse device

SEE ALSO

TOC

cua(4), bm(4)

TOC

NAME

null, zero - data sink

DESCRIPTION

Data written on a **null** or **zero** special file is discarded.

Reads from the **null** special file always return end of file, whereas reads from **zero** always return \0 characters.

null and **zero** are typically created by:

```
mknod -m 666 /dev/null c 1 3
mknod -m 666 /dev/zero c 1 5
chown root.mem /dev/null /dev/zero
```

NOTES

If these devices are not writable and readable for all users, many programs will act strange.

FILES

/dev/null
/dev/zero

SEE ALSO

mknod(1), **chown**(1)

NAME

ram - ram disk device

DESCRIPTION

Ram is a block device to access the ram disk in raw mode.

It is typically created by:

```
mknod -m 660 /dev/ram b 1 1
chown root.disk /dev/ram
```

FILES

/dev/ram

SEE ALSO

mknod(1), **chown**(1), **mount**(8)


TOC

TOC

NAME

random, urandom - kernel random number source devices

DESCRIPTION

The character special files **/dev/random** and **/dev/urandom** (present since Linux 1.3.30) provide an interface to the kernel's random number generator. File **/dev/random** has major device number 1 and minor device number 8. File **/dev/urandom** has major device number 1 and minor device number 9.

The random number generator gathers environmental noise from device drivers and other sources into an entropy pool. The generator also keeps an estimate of the number of bit of the noise in the entropy pool. From this entropy pool random numbers are created.

When read, the **/dev/random** device will only return random bytes within the estimated number of bits of noise in the entropy pool. **/dev/random** should be suitable for uses that need very high quality randomness such as one-time pad or key generation. When the entropy pool is empty, reads to **/dev/random** will block until additional environmental noise is gathered.

When read, **/dev/urandom** device will return as many bytes as are requested. As a result, if there is not sufficient entropy in the entropy pool, the returned values are theoretically vulnerable to a cryptographic attack on the algorithms used by the driver. Knowledge of how to do this is not available in the current non-classified literature, but it is theoretically possible that such an attack may exist. If this is a concern in your application, use **/dev/random** instead.

CONFIGURING

If your system does not have **/dev/random** and **/dev/urandom** created already, they can be created with the following commands:

```
mknod -m 644 /dev/random c 1 8
mknod -m 644 /dev/urandom c 1 9
chown root.root /dev/random /dev/urandom
```

When a Linux system starts up without much operator interaction, the entropy pool may be in a fairly predictable state. This reduces the actual amount of noise in the entropy pool below the estimate. In order to counteract this effect, it helps to carry entropy pool information across shut-downs and start-ups. To do this, add the following lines to an appropriate script which is run during the Linux system start-up sequence:

```
echo "Initializing kernel random number generator..."
# Initialize kernel random number generator with random seed
# from last shut-down (or start-up) to this start-up.  Load and
# then save 512 bytes, which is the size of the entropy pool.
if [ -f /var/random-seed ]; then
     cat /var/random-seed >/dev/urandom
fi
dd if=/dev/urandom of=/var/random-seed count=1
```

Also, add the following lines in an appropriate script which is run during the Linux system shutdown:

```
# Carry a random seed from shut-down to start-up for the random
# number generator.  Save 512 bytes, which is the size of the
# random number generator's entropy pool.
echo "Saving random seed..."
dd if=/dev/urandom of=/var/random-seed count=1
```

FILES

/dev/random
/dev/urandom

AUTHOR

The kernel's random number generator was written by Theodore Ts'o (tytso@athena.mit.edu).

SEE ALSO

mknod (1)
RFC 1750, "Randomness Recommendations for Security"

NAME

sd - Driver for SCSI Disk Drives

SYNOPSIS

```
#include <linux/hdreg.h>
```

CONFIG

The block device name has the following form: **sd***lp*, where **l** is a letter denoting the physical drive, and **p** is a number denoting the partition on that physical drive. Often, the partition number, **p**, will be left off when the device corresponds to the whole drive.

SCSI disks have a major device number of 8, and a minor device number of the form (16 * *drive_number)* + *partition_number*, where *drive_number* is the number of the physical drive in order of detection, and *partition_number* is as follows:

partition 0 is the whole drive
partitions 1-4 are the DOS "primary" partitions
partitions 5-8 are the DOS "extended" (or "logical") partitions

For example, **/dev/sda** will have major 8, minor 0, and will refer to all of the first SCSI drive in the system; and **/dev/sdb3** will have major 8, minor 19, and will refer to the third DOS "primary" partition on the second SCSI drive in the system.

At this time, only block devices are provided. Raw devices

have not yet been implemented.

DESCRIPTION

The following ioctl's are provided:

HDIO_REQ
Returns the BIOS disk parameters in the following structure:

```
struct hd_geometry {
      unsigned char heads;
      unsigned char sectors;
      unsigned short cylinders;
      unsigned long start;
};
```

A pointer to this structure is passed as the **ioctl**(2) parameter.

The information returned in the parameter is the disk geometry of the drive *as understood by DOS*! This geometry is *not* the physical geometry of the drive. It is used when constructing the drive's partition table, however, and is needed for convenient operation of **fdisk**(1),**efdisk**(1), and **lilo**(1). If the geometry information is not available, zero will be returned for all of the parameters.

BLKGETSIZE
Returns the device size in sectors. The **ioctl**(2) parameter should be a pointer to a **long**.

BLKRRPART
Forces a re-read of the SCSI disk partition tables. No parameter is needed.

The **scsi**(4) ioctls are also supported. If the **ioctl**(2) parameter is required, and it is NULL, then **ioctl**(2) will return -EINVAL.

FILES


```
/dev/sd[a-h]: the whole device
/dev/sd[a-h][0-8]: individual block partitions
```

SEE ALSO

scsi(4)

NAME

st - SCSI tape device

SYNOPSIS

```
#include <sys/mtio.h>

int ioctl(int fd, int request [, (void *)arg3
int ioctl(int fd, MTIOCTOP, (struct mtop *)mt_cmd))
int ioctl(int fd, MTIOCGET, (struct mtget *)mt_status))
int ioctl(int fd, MTIOCPOS, (struct mtpos *)mt_pos))
```

DESCRIPTION

The **st** driver provides the interface to a variety of SCSI tape devices. Currently, the driver takes control of all detected devices of type sequential-access. The **st** driver uses major device number 9.

Each device uses two minor device numbers: a principal minor device number, **n**, assigned sequentially in order of detection, and a no-rewind device number, Devices opened using the principal device number will be sent a REWIND command when they are closed. Devices opened using the no-rewind device number will not. Options such as density or block size are not coded in the minor device number. These options must be set by explicit **ioctl()** calls and remain in effect when the device is closed and reopened.

Devices are typically created by:

```
mknod -m 660 /dev/st0 c 9 0
mknod -m 660 /dev/st1 c 9 1
mknod -m 660 /dev/nst0 c 9 128
```


```
mknod -m 660 /dev/nst1 c 9 129
```

There is no corresponding block device. The character device provides buffering and read-ahead by default and supports reads and writes of arbitrary size (limited by the drivers internal buffer size, which defaults to 32768 bytes, but can be changed either by using a kernel startup option or by changing a compile-time constant).

Device **/dev/tape** is usually created as a hard or soft link to the default tape device on the system.

IOCTLS

The driver supports three ioctl requests. Requests not recognized by the **st** driver are passed to the **scsi** driver. The definitions below are from <*linux/mtio.h*>:

MTIOCTOP - Perform a tape operation

This request takes an argument of type **(struct mtop *)**. Not all drives support all operations. The driver returns an EIO error if the drive rejects an operation.

```
/* Structure for MTIOCTOP - mag tape op command: */
struct mtop {
    short  mt_op;    /* operations defined below */
    int    mt_count; /* how many of them */
};
```

Magnetic Tape operations:

MTBSF	Backward space over **mt_count** filemarks.
MTBSFM	Backward space over **mt_count** filemarks. Reposition the tape to the EOT side of the last filemark.
MTBSR	Backward space over **mt_count** records (tape blocks).
MTBSS	Backward space over **mt_count** setmarks.
MTEOM	Go to the end of the recorded media (for appending files).
MTERASE	Erase tape.
MTFSF	Forward space over **mt_count** filemarks.
MTFSFM	Forward space over **mt_count** filemarks. Reposition the tape to the BOT side of the last

filemark.

MTFSR Forward space over **mt_count** records (tape blocks).

MTFSS Forward space over **mt_count** setmarks.

MTNOP No op - flushes the drivers buffer as a side effect. Should be used before reading status with MTIOCGET.

MTOFFL Rewind and put the drive off line.

MTRESET Reset drive.

MTRETEN Retension tape.

MTREW Rewind.

MTSEEK Seek to the tape block number specified in **mt_count**. This operation requires either a SCSI-2 drive that supports the LOCATE command (device-specific address) or a Tandberg-compatible SCSI-1 drive (Tandberg, Archive Viper, Wangtek, ...). The block number should be one that was previously returned by MTIOCPOS because the number is device-specific.

MTSETBLK Set the drives block length to the value specified in **mt_count**. A block length of zero sets the drive to variable block size mode.

MTSETDENSITY Set the tape density to the code in **mt_count**. Some useful density codes are:

0x00 Implicit	0x11 QIC-525
0x04 QIC-11	0x12 QIC-1350
0x05 QIC-24	0x13 DDS
0x0F QIC-120	0x14 Exabyte EXB-8200
0x10 QIC-150	0x15 Exabyte EXB-8500

MTWEOF Write **mt_count** filemarks.

MTWSM Write **mt_count** setmarks.

MTSETDRVBUFFER

Set various drive and driver options according to bits encoded in **mt_count**. These consist of the drives buffering mode, 6 Boolean driver options and the buffer write threshold. These parameters are initialized only when the device is first detected. The settings persist when the device is closed and reopened. A single operation can affect (a) just the buffering mode, (b) just the Boolean options, or (c) just the write threshold.

A value having zeros in the high-order 4 bits will be used to set the drives buffering mode. The buffering modes are:

0 The drive will not report GOOD status on write commands until the data blocks are actually written to the medium.

1 The drive may report GOOD status on write commands as soon as all the data has been transferred to the drives internal buffer.

2 The drive may report GOOD status on write commands as soon as (a) all the data has been transferred to the drives internal buffer, and (b) all buffered data from different initiators has been successfully written to the medium.

To control the write threshold the value in **mt_count** must include the constant MT_ST_WRITE_THRESHOLD logically ORed with a block count in the low 28 bits. The block count refers to 1024-byte blocks, not the physical block size on the tape. The threshold cannot exceed the drivers internal buffer size (see **DESCRIPTION**, above).

To set and clear the Boolean options the value in **mt_count** must include the constant MT_ST_BOOLEANS logically ORed with whatever combination of the following options is desired. Any options not specified will be set false. The Boolean options are:

MT_ST_BUFFER_WRITES (Default: true)
Buffer all write operations. If this option is false and the drive uses a fixed block size, then all write operations must be for a multiple of the block size. This option must be set false to write reliable multi-volume archives.

MT_ST_ASYNC_WRITES (Default: true)
When this options is true write operations return immediately without waiting for the data to be transferred to the drive if the data fits into the drivers buffer. The write threshold determines how full the buffer must be before a new SCSI write command is issued. Any errors reported by the drive will be held until the next operation. This option must be set false to write reliable multi-volume archives.

MT_ST_READ_AHEAD (Default: true)

This option causes the driver to provide read buffering and read-ahead. If this option is false and the drive uses a fixed block size, then all read operations must be for a multiple of the block size.

MT_ST_TWO_FM (Default: false)

This option modifies the driver behavior when a file is closed. The normal action is to write a single filemark. If the option is true the driver will write two filemarks and backspace over the second one.

Note: This option should not be set true for QIC tape drives since they are unable to overwrite a filemark. These drives detect the end of recorded data by testing for blank tape rather than two consecutive filemarks.

MT_ST_DEBUGGING (Default: false)

This option turns on various debugging messages from the driver (effective only if the driver was compiled with DEBUG defined).

MT_ST_FAST_EOM (Default: false)

This option causes the MTEOM operation to be sent directly to the drive, potentially speeding up the operation but causing the driver to lose track of the current file number normally returned by the MTIOCGET request. If MT_ST_FAST_EOM is false the driver will respond to an MTEOM request by forward spacing over files.

EXAMPLE

```
struct mtop mt_cmd;
mt_cmd.mt_op = MTSETDRVBUFFER;
mt_cmd.mt_count = MT_ST_BOOLEANS |
          MT_ST_BUFFER_WRITES |
          MT_ST_ASYNC_WRITES;
ioctl(fd, MTIOCTOP, &mt_cmd));
```

MTIOCGET - Get status

This request takes an argument of type **(struct mtget *)**. The driver returns an EIO error if the drive rejects an operation.

```
/* structure for MTIOCGET - mag tape get status command */
struct mtget {
```


```
    long   mt_type;
    long   mt_resid;
    /* the following registers are device dependent */
    long   mt_dsreg;
    long   mt_gstat;
    long   mt_erreg;
    /* The next two fields are not always used */
    daddr_t          mt_fileno;
    daddr_t          mt_blkno;
};
```

mt_type The header file defines many values for **mt_type**, but the current driver reports only the generic types MT_ISSCSI1 (Generic SCSI-1 tape) and MT_ISSCSI2 (Generic SCSI-2 tape).

mt_resid is always zero. (Not implemented for SCSI tape drives.)

mt_dsreg reports the drives current settings for block size (in the low 24 bits) and density (in the high 8 bits). These fields are defined by MT_ST_BLKSIZE_SHIFT, MT_ST_BLKSIZE_MASK, MT_ST_DENSITY_SHIFT, and MT_ST_DENSITY_MASK.

mt_gstat reports generic (device independent) status information. The header file defines macros for testing these status bits:

GMT_EOF(**x**): The tape is positioned just after a filemark (always false after an MTSEEK operation).

GMT_BOT(**x**): The tape is positioned at the beginning of the first file (always false after an MTSEEK operation).

GMT_EOT(**x**): A tape operation has reached the physical End Of Tape.

GMT_SM(**x**): The tape is currently positioned at a setmark (always false after an MTSEEK operation).

GMT_EOD(**x**): The tape is positioned at the end of recorded data.

GMT_WR_PROT(**x**): The drive is write-protected. For some drives this can also mean that the drive does not support writing on the current medium type.

GMT_ONLINE(**x**): The last **open()** found the drive with a tape in place and ready for operation.

GMT_D_6250(**x**), GMT_D_1600(**x**), GMT_D_800(**x**): This generic status information reports the

current density setting for 9-track 1/2" tape drives only.

GMT_DR_OPEN(**x**): The drive does not have a tape in place.

GMT_IM_REP_EN(**x**): Immediate report mode (not supported).

mt_erreg The only field defined in **mt_erreg** is the recovered error count in the low 16 bits (as defined by MT_ST_SOFTERR_SHIFT and MT_ST_SOFTERR_MASK). Due to inconsistencies in the way drives report recovered errors, this count is often not maintained.

mt_fileno reports the current file number (zero-based). This value is set to -1 when the file number is unknown (e.g., after MTBSS or MTSEEK).

mt_blkno reports the block number (zero-based) within the current file. This value is set to -1 when the block number is unknown (e.g., after MTBSF, MTBSS, or MTSEEK).

MTIOCPOS - Get tape position

This request takes an argument of type **(struct mtpos *)** and reports the drives notion of the current tape block number, which is not the same as **mt_blkno** returned by MTIOCGET. This drive must be a SCSI-2 drive that supports the READ POSITION command (device-specific address) or a Tandberg-compatible SCSI-1 drive (Tandberg, Archive Viper, Wangtek, ...).

```
/* structure for MTIOCPOS - mag tape get position command */
struct     mtpos {
    long   mt_blkno; /* current block number */
};
```

RETURN VALUE

EIO The requested operation could not be completed.

ENOSPC A write operation could not be completed

because the tape reached end-of-medium.

EACCES — An attempt was made to write or erase a write-protected tape. (This error is not detected during **open()**.)

ENXIO — During opening, the tape device does not exist.

EBUSY — The device is already in use or the driver was unable to allocate a buffer.

EOVERFLOW — An attempt was made to read or write a variable-length block that is larger than the drivers internal buffer.

EINVAL — An **ioctl()** had an illegal argument, or a requested block size was illegal.

ENOSYS — Unknown **ioctl()**.

COPYRIGHT

SEE ALSO

mt(1)

TOC

NAME

tty - controlling terminal

DESCRIPTION

The file **/dev/tty** is a character file with major number 5 and minor number 0, usually of mode 0666 and owner.group root.tty. It is a synonym for the controlling terminal of a process, if any.

In addition to the **ioctl()** requests supported by the device that **tty** refers to, the following **ioctl()** request is supported:

TIOCNOTTY
Detach the current process from its controlling terminal, and remove it from its current process group, without attaching it to a new process group (that is, set its process group ID to zero). This **ioctl()** call only works on file descriptors connected to **/dev/tty**; this is used by daemon processes when they are invoked by a user at a terminal. The process attempts to open **/dev/tty**; if the open succeeds, it detaches itself from the terminal by using **TIOCNOTTY**, while if the open fails, it is obviously not attached to a terminal and does not need to detach itself.

FILES

/dev/tty

SEE ALSO

mknod(1), **chown**(1), **getty**(1), **console**(4), **ttys**(4)

NAME

ttys - serial terminal lines

DESCRIPTION

ttyS[0-3] are character devices for the serial terminal lines.

They are typically created by:

```
mknod -m 660 /dev/ttyS0 c 4 64 # base address 0x03f8
mknod -m 660 /dev/ttyS1 c 4 65 # base address 0x02f8
mknod -m 660 /dev/ttyS2 c 4 66 # base address 0x03e8
mknod -m 660 /dev/ttyS3 c 4 67 # base address 0x02e8
chown root.tty /dev/ttyS[0-3]
```

FILES

/dev/ttyS[0-3]

SEE ALSO

mknod(1), **chown**(1), **getty**(1),

TOC

NAME

vcs, vcsa - virtual console memory

DESCRIPTION

/dev/vcs0 is a character device with major number 7 and minor number 0, usually of mode 0644 and owner root.tty. It refers to the memory of the currently displayed virtual console terminal.

/dev/vcs[1-63] are character devices for virtual console terminals, they have major number 7 and minor number 1 to 63, usually mode 0644 and owner root.tty. **/dev/vcsa[0-63]** are the same, but including attributes, and prefixed with four bytes giving the screen dimensions and cursor position: *lines*, *columns*, **x**, **y**. (**x** = **y** = 0 at the top left corner of the screen.)

These replace the screendump ioctls of **console**(4), so the system administrator can control access using file system permissions.

The devices for the first eight virtual consoles may be created by:

```
for x in 0 1 2 3 4 5 6 7 8; do
     mknod -m 644 /dev/vcs$x c 7 $x;
     mknod -m 644 /dev/vcsa$x c 7 $[$x+128];
done
chown root.tty /dev/vcs*
```

No **ioctl()** requests are supported.

EXAMPLES

You may do a screendump on vt3 by switching to vt1 and typing *cat /dev/vcs3 >foo*. Note that the output does not contain newline characters, so some processing may be required, like in *fold* **-w** *81 /dev/vcs3* | *lpr* or (horrors) *setterm -dump* **3** *-file /proc/self/fd/***1**.

This program displays the character and screen attributes under the cursor of the second virtual console, then changes the background color there:

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>

void main()
{    int fd;
     struct {char lines, cols, x, y;} scrn;
     char ch, attrib;

     fd = open("/dev/vcsa2", O_RDWR);
     (void)read(fd, &scrn, 4);
     (void)lseek(fd, 4 + 2*(scrn.y*scrn.cols + scrn.x), 0);
     (void)read(fd, &ch, 1);
     (void)read(fd, &attrib, 1);
     printf("ch='%c' attrib=0x%02x\n", ch, attrib);
     attrib ^= 0x10;
     (void)lseek(fd, -1, 1);
     (void)write(fd, &attrib, 1);
}
```

FILES

```
/dev/vcs[0-63]
/dev/vcsa[0-63]
```

AUTHOR

Andries Brouwer <aeb@cwi.nl>

HISTORY

Introduced with version 1.1.92 of the Linux kernel.

SEE ALSO

console(4), **tty**(4), **ttys**(4),

TOC

NAME

wavelan - AT&T GIS WaveLAN ISA device driver

SYNOPSIS

insmod wavelan_cs.o [io=B,B..] [irq=I,I..] [name=N,N..

DESCRIPTION

wavelan is the low-level device driver for the NCR / AT&T / Lucent **WaveLAN ISA** and Digital (DEC) **RoamAbout DS** wireless ethernet adapter. This driver is available as a module or might be compiled in the kernel. This driver supports multiple cards in both forms (up to 4) and allocates the next available ethernet device (eth0..eth#) for each card found, unless a device name is explicitely specified (see below). This device name will be reported in the kernel log file with the MAC address, NWID and frequency used by the card.

PARAMETERS

This section apply to the module form (parameters passed on the *insmod*(8) command line). If the driver is included in the kernel, use the *ether=IRQ,IO,NAME* syntax on the kernel command line.

io Specify the list of base address where to search for wavelan cards (setting by dip switch on the card). If you don't specify any io address, the driver will scan


TOC

0x390 and 0x3E0 addresses, which might conflict with other hardware...

irq Set the list of irq that each wavelan card should use (the value is saved in permanent storage for future use).

name Set the list of name to be used for each wavelan cards device (name used by *ifconfig*(8)).

WIRELESS EXTENSIONS

Use *iwconfig*(8) to manipulate wireless extensions.

NWID (or domain)
Set the network ID [**0** to *FFFF*] or disable it [*off*]. As the NWID is stored in the card Permanent Storage Area, it will be reuse at any further invocation of the driver.

Frequency & channels
For the 2.4GHz 2.00 Hardware, you are able to set the frequency by specifying one of the 10 defined channels (*2.412*, *2.422*, *2.425*, *2.4305*, *2.432*, *2.442*, *2.452*, or *2.484)* or directly by its value. The frequency is changed immediately and permanentely. Frequency availability depend on the regulations...

Statistics spy
Set a list of MAC addresses in the driver (up to 8) and get the last quality of link for each of those (see *iwspy*(8)).

/proc/net/wireless
status is the status reported by the modem. *Link quality* reports the quality of the modulation on the air (direct sequence spread spectrum) [max = 16]. *Level* and *Noise* refer to the signal level and noise level [max = 64]. The *crypt discarded packet* and *misc discarded packet* counters are not implemented.

PRIVATE IOCTL

You may use *iwpriv*(8) to manipulate private ioctls.

Quality and Level threshold
Enable you the define the quality and level threshold used by the modem (packet below that level are discarded).

Histogram
This functionality allow to set a number of signal level intervals and to count the number of packets received in each of those defined intervals. This distribution might be used to calculate the mean value and standard deviation of the signal level.

SPECIFIC NOTES

This driver will fail to detect some **non NCR/ATT&T/Lucent** Wavelan cards. If it's your case, you must look in the source code on how to add your card to the detection routine.

Some of the mentioned features are optional. You may enable to disable them by changing flags in the driver header and recompile.

AUTHOR

Bruce Janson - bruce@cs.usyd.edu.au
Jean Tourrilhes - jt@hplb.hpl.hp.com
(+ others - see source code for details)

SEE ALSO

wavelan_cs(4), **ifconfig**(8), **insmod**(8), **iwconfig**(8), **iwspy**(8), **iwpriv**(8).

Linux Man Pages Section 5

- charmap.5
- environ.5
- fs.5
- group.5
- intro.5
- ipc.5
- issue.5
- lilo.conf
- locale.5
- motd.5
- nologin.5
- nsswitch.5
- passwd.5
- proc.5
- protocols.5
- securetty.5
- services.5
- shells.5
- termcap.5
- ttytype.5
- utmp.5
- wtmp.5

TOC

NAME

charmap - character symbols to define character encodings

DESCRIPTION

A character set description (charmap) defines a characterset of available characters and their encodings. All supported character sets should have the **portable character set** as a proper subset. The portable character set is defined in the file */usr/lib/nls/charmap/POSIX* for reference purposes.

SYNTAX

The charmap file starts with a header, that may consist of the following keywords:

<codeset>
is followed by the name of the codeset.

<mb_cur_max>
is followed by the max number of bytes for a multibyte-character. Multibyte characters are currently not supported. The default value is 1.

<mb_cur_min>
is followed by the min number of bytes for a character. This value must be less or equal than **mb_cur_max.** If not specified, it defaults to **mb_cur_max.**

<escape_char>
is followed by a character that should be used as the escape-character for the rest of the file to mark characters that should be interpreted in a special way. It

defaults to the backslash (\ ).

<comment_char>
is followed by a character that will be used as the comment-character for the rest of the file. It defaults to the number sign (#).

The charmap-definition itself starts with the keyword **CHARMAP** in column 1.

The following lines may have one of the two following forms to define the character-encodings:

<symbolic-name> *<encoding>* *<comments>*
This for defines exactly one character and its encoding.

<symbolic-name>...*<symbolic-name>* *<encoding>* *<comments>*
This form defines a couple of characters. This is only useful for mutlibyte-characters, which are currently not implemented.

The last line in a charmap-definition file must contain **END CHARMAP.**

SYMBOLIC NAMES

A **symbolic name** for a character contains only characters of the **portable character set.** The name itself isenclosed between angle brackets. Characters following the **<escape_char>** are interpreted as itself; for example, the sequence **'<\\\>>'** represents the symbolic name **'\>'** enclosed in angle brackets.

CHARACTER ENCODING

The encoding may be in each of the following three forms:

<*escape_char*>**d**<*number*>
with a decimal number

<*escape_char*>**x**<*number*>
with a hexadecimal number

<*escape_char*><*number*>
with an octal number.

FILES

*/usr/lib/nls/charmap/**

AUTHOR

Jochen Hein (jochen.hein@delphi.central.de)

CONFORMING TO

POSIX.2

SEE ALSO

setlocale(3), **localeconv**(3), **locale**(1), **locale**(5), **localedef**(1),

TOC

NAME

environ - user environment

SYNOPSIS

extern char **environ;

DESCRIPTION

The variable *environ* points to an array of strings called the `environment'. (This variable must be declared in the user program, but is declared in the header file *unistd.h* in case the header files came from libc4 or libc5, and in case they came from glibc and **_GNU_SOURCE** was defined.) This array of strings is made available to the process by the **exec**(2) call that started the process. By convention these strings have the form `*name*=*value*'. Common examples are:

USER The name of the logged-in user (used by some BSD-derived programs).

LOGNAME
The name of the logged-in user (used by some System-V derived programs).

HOME A user's login directory, set by **login**(1) from the password file **passwd**(5).

LANG The name of a locale to use for locale categories when not overridden by **LC_ALL** or more specific environment variables.

PATH The sequence of directory prefixes that **sh**(1) and many

other programs apply in searching for a file known by an incomplete path name. The prefixes are separated by `:'. (Similarly one has **CDPATH** used by some shells to find the target of a change directory command, **MANPATH** used by **man**(1) to find manual pages, etc.)

PWD The current working directory. Set by some shells.

SHELL
The file name of the user's login shell.

TERM The terminal type for which output is to be prepared.

Further names may be placed in the environment by the **export** command and `name=value' in **sh**(1), or by the **setenv** command if you use **csh**(1). Arguments may also be placed in the environment at the point of an **exec**(2). A C program can manipulate its environment using the functions **getenv**(), **putenv**(), **setenv**() and **unsetenv**().

Note that the behaviour of many programs and library routines is influenced by the presence or value of certain environment variables. A random collection:

The variables LANG, LANGUAGE, NLSPATH, LOCPATH, LC_ALL, LC_MESSAGES etc. influence locale handling.

TMPDIR influences the path prefix of names created by **tmpnam()** and other routines, the temporary directory used by **sort**(1) and other programs, etc.

LD_LIBRARY_PATH, LD_PRELOAD and other LD_* variables influence the behaviour of the dynamic loader/linker.

POSIXLY_CORRECT makes certain programs and library routines follow the prescriptions of POSIX.

The behaviour of **malloc**() is influenced by MALLOC_* variables.

The variable HOSTALIASES gives the name of a file containing aliases to be used with **gethostbyname**().

TZ and TZDIR give time zone information.

TERMCAP gives information on how to address a given terminal

(or gives the name of a file containing such information).

Etc. etc.

Clearly there is a security risk here. Many a system command has been tricked into mischief by a user who specified unusual values for IFS or LD_LIBRARY_PATH.

SEE ALSO

login(1), **sh**(1), **bash**(1), **csh**(1), **tcsh**(1), **exec**(2), **getenv**(3), **putenv**(3), **setenv**(3), **unsetenv**(3).

TOC

NAME

filesystems - Linux filesystem types: minix, ext, ext2, xia, msdos, umsdos, vfat, proc, nfs, iso9660, hpfs, sysv, smb, ncpfs

DESCRIPTION

In the file */proc/filesystems* you can find which filesystems your kernel currently supports. (If you need a currently unsupported one, insert the corresponding module or recompile the kernel.)

Below a description of the various filesystems.

minix
is the filesystem used in the Minix operating system, the first to run under Linux. It has a number of shortcomings: a 64MB partition size limit, short filenames, a single time stamp, etc.

It remains useful for floppies and RAM disks.

ext is an elaborate extension of the **minix** filesystem. It has been completely superseded by the second version of the extended filesystem (**ext2**) and will eventually be removed from the kernel.

ext2 is the high performance disk filesystem used by Linux for fixed disks as well as removable media.

The second extended filesystem was designed as an extension of the extended file system (**ext**). **ext2** offers the best performance (in terms of speed and CPU usage) of the filesystems supported under Linux.

xiafs
was designed and implemented to be a stable, safe filesystem by extending the Minix filesystem code. It provides the basic most requested features without undue complexity.

The **xia** filesystem is no longer actively developed or maintained. It is used infrequently.

msdos
is the filesystem used by DOS, Windows, and some OS/2 computers. **msdos** filenames can be no longer than an 8 character name followed by an optional period and 3 character extension.

umsdos
is an extended DOS filesystem used by Linux. It adds capability for long filenames, UID/GID, POSIX permissions, and special files (devices, named pipes, etc.) under the DOS filesystem, without sacrificing compatibility with DOS.

vfat is extended DOS filesystem used by Microsoft Windows95 and Windows NT. VFAT adds capability for long filenames under the MSDOS filesystem.

proc is a pseudo-filesystem which is used as an interface to kernel data structures rather than reading and interpreting */dev/kmem*. In particular, its files do not take disk space. See proc(5).

iso9660
is a CD-ROM filesystem type conforming to the ISO 9660 standard.

High Sierra
Linux supports High Sierra, the precursor to the ISO 9660 standard for CD-ROM filesystems. It is automatically recognized within the **iso9660** filesystem support under Linux.

Rock Ridge
Linux also supports the System Use Sharing Protocol records specified by the Rock Ridge Interchange Protocol. They are used to further describe the files in the **iso9660** filesystem to a

UNIX host, and provides information such as long filenames, UID/GID, POSIX permissions, and devices. It is automatically recognized within the **iso9660** filesystem support under Linux.

hpfs is the High Performance Filesystem, used in OS/2. This filesystem is read-only under Linux due to the lack of available documentation.

sysv is an implementation of the SystemV/Coherent filesystem for Linux. It implements all of Xenix FS, SystemV/386 FS, and Coherent FS.

nfs is the network filesystem used to access disks located on remote computers.

smb is a network filesystem that supports the SMB protocol, used by Windows for Workgroups, Windows NT, and Lan Manager.

To use **smb** fs, you need a special mount program, which can be found in the ksmbfs package, found at *ftp://sunsite.unc.edu/pub/Linux/system/Filesystems/smbfs*.

ncpfs
is a network filesystem that supports the NCP protocol, used by Novell NetWare.

To use **ncpfs**, you need special programs, which can be found at *ftp://linux01.gwdg.de/pub/ncpfs*.

SEE ALSO

proc(5), **fsck**(8), **mkfs**(8), **mount**(8).

TOC

NAME

group - user group file

DESCRIPTION

/etc/group is an ASCII file which defines the groups to which users belong. There is one entry per line, and each line has the format:

```
group_name:passwd:GID:user_list
```

The field descriptions are:

group_name
: the name of the group.

password
: the (encrypted) group password. If this field is empty, no password is needed.

GID
: the numerical group ID.

user_list
: all the group member's user names, separated by commas.

FILES

/etc/group

SEE ALSO

login(1), **newgrp**(1), **passwd**(5)

NAME

intro - Introduction to file formats

DESCRIPTION

This chapter describes various file formats and protocols, and the used C structures, if any.

AUTHORS

Look at the header of the manual page for the author(s) and copyright conditions. Note that these can be different from page to page!


TOC

TOC

NAME

ipc - System V interprocess communication mechanisms

SYNOPSIS

```
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>
# include <sys/sem.h>
# include <sys/shm.h>
```

DESCRIPTION

The manual page refers to the Linux implementation of the System V interprocess communication mechanisms: message queues, semaphore sets and shared memory segments. In the following, the word **resource** means an instantiation of one among such mechanisms.

Resource Access Permissions

For each resource the system uses a common structure of type **struct ipc_perm** to store information needed in determining permissions to perform an ipc operation. The **ipc_perm** structure, defined by the *<sys/ipc.h>* system header file, includes the following members:

```
ushort cuid;    /* creator user id */
ushort cgid;    /* creator group id */
ushort uid; /* owner user id */
ushort gid; /* owner group id */
ushort mode; /* r/w permissions */
```

The **mode** member of the **ipc_perm** structure defines, with its

lower 9 bits, the access permissions to the resource for a process executing an ipc system call. The permissions are interpreted as follows:

```
     0400 Read by user.
     0200 Write by user.
9     0040 Read by group.
     0020 Write by group.
9     0004 Read by others.
     0002 Write by others.
```

Bits 0100, 0010 and 0001 (the execute bits) are unused by the system. Furthermore "write" effectively means "alter" for a semaphore set.

The same system header file defines also the following symbolic constants:

IPC_CREAT Create entry if key doesn't exists.

IPC_NOWAIT Error if request must wait.

IPC_PRIVATE Private key.

IPC_RMID Remove resource.

IPC_SET Set resource options.

IPC_STAT Get resource options.

Note that **IPC_PRIVATE** is a **key_t** type, while all the others symbolic constants are flag fields or-able into an **int** type variable.

Message Queues

A message queue is uniquely identified by a positive integer (its *msqid*) and has an associated data structure of type **struct msquid_ds**, defined in *<sys/msg.h>*, containing the following members:

```
     struct ipc_perm msg_perm;
     ushort msg_qnum;     /* no of messages on queue */
     ushort msg_qbytes;   /* bytes max on a queue */
     ushort msg_lspid;    /* pid of last msgsnd call */
```


```
    ushort msg_lrpid;    /* pid of last msgrcv call */
    time_t msg_stime;    /* last msgsnd time */
    time_t msg_rtime;    /* last msgrcv time */
    time_t msg_ctime;    /* last change time */
```

msg_perm **ipc_perm** structure that specifies the access permissions on the message queue.

msg_qnum Number of messages currently on the message queue.

msg_qbytes Maximum number of bytes of message text allowed on the message queue.

msg_lspid ID of the process that performed the last **msgsnd** system call.

msg_lrpid ID of the process that performed the last **msgrcv** system call.

msg_stime Time of the last **msgsnd** system call.

msg_rtime Time of the last **msgcv** system call.

msg_ctime Time of the last system call that changed a member of the **msqid_ds** structure.

A semaphore set is uniquely identified by a positive integer (its *semid)* and has an associated data structure of type **struct semid_ds**, defined in <*sys/sem.h*>, containing the following members:

```
    struct ipc_perm sem_perm;
    time_t sem_otime;    /* last operation time */
    time_t sem_ctime;    /* last change time */
    ushort sem_nsems;    /* count of sems in set */
```

sem_perm **ipc_perm** structure that specifies the access permissions on the semaphore set.

sem_otime Time of last **semop** system call.

sem_ctime Time of last **semctl** system call that changed a member of the above structure or of one semaphore belonging to the set.

sem_nsems Number of semaphores in the set. Each semaphore of the set is referenced by a non-negative integer ranging from **0** to **sem_nsems-1**.

A semaphore is a data structure of type **struct sem** containing the following members:

```
ushort semval;  /* semaphore value */
short sempid;   /* pid for last operation */
ushort semncnt;       /*  no.  of  awaiting  semval  to
increase */
ushort semzcnt;      /* no. of awaiting semval = 0 */
```

semval Semaphore value: a non-negative integer.

sempid ID of the last process that performed a semaphore operation on this semaphore.

semncnt Number of processes suspended awaiting for **semval** to increase.

semznt Number of processes suspended awaiting for **semval** to become zero.

Shared Memory Segments

A shared memory segment is uniquely identified by a positive integer (its *shmid)* and has an associated data structure of type **struct shmid_ds**, defined in <*sys/shm.h*>, containing the following members:

```
struct ipc_perm shm_perm;
int shm_segsz;  /* size of segment */

ushort shm_lpid;     /* pid, last operation */
short shm_nattch;    /* no. of current attaches */
time_t shm_atime;    /* time of last attach */
time_t shm_dtime;    /* time of last detach */
time_t shm_ctime;    /* time of last change */
```

shm_perm **ipc_perm** structure that specifies the access permissions on the shared memory segment.

shm_segsz Size in bytes of the shared memory segment.

shm_cpid ID of the process that created the shared memory segment.

shm_lpid ID of the last process that executed a **shmat** or **shmdt** system call.

shm_nattch Number of current alive attaches for this shared memory segment.

shm_atime Time of the last **shmat** system call.

shm_dtime Time of the last **shmdt** system call.

shm_ctime Time of the last **shmctl** system call that changed **shmid_ds**.

SEE ALSO

ftok(3), **msgctl**(2), **msgget**(2), **msgrcv**(2), **msgsnd**(2), **semctl**(2), **semget**(2), **semop**(2), **shmat**(2), **shmctl**(2), **shmget**(2), **shmdt (2).**

NAME

issue - pre-login message and identification file

DESCRIPTION

The file **/etc/issue** is an text file which contains a message or system identification to be printed before the login prompt. It may contain various *@char* and *\char* sequences, if supported by **getty**(1).

FILES

/etc/issue

SEE ALSO

getty(1), **motd**(5)

TOC

NAME

lilo.conf - configuration file for lilo

DESCRIPTION

This file, by default */etc/lilo.conf*, is read by the boot loader installer lilo (see lilo(8)).

It might look as follows:

```
boot = /dev/hda
delay = 40
compact
vga = normal
root = /dev/hda1
read-only
image = /zImage-1.5.99
        label = try
image = /zImage-1.0.9
        label = 1.0.9
image = /tamu/vmlinuz
     label = tamu
     root = /dev/hdb2
     vga = ask
other = /dev/hda3
     label = dos
     table = /dev/hda
```

This configuration file specifies that lilo uses the Master Boot Record on /dev/hda. (For a discussion of the various ways to use lilo, and the interaction with other operating systems, see user.tex from the lilo documentation.)

When booting, the boot loader will wait four seconds (40 deciseconds) for you to press Shift. If you don't, then the


TOC

first kernel image mentioned (/zImage-1.5.99, that you probably installed just five minutes ago) will be booted. If you do, the boot loader will ask you which image to boot. In case you forgot the possible choices, press [TAB] (or [?], if you have a US keyboard), and you will be presented with a menu. You now have the choice of booting this brand-new kernel, or an old trusted kernel, or a kernel on another root file system (just in case you did something stupid on your usual rootfs), or booting a different operating system. There can be up to 16 images mentioned in lilo.conf.

As can be seen above, a configuration file starts with a number of global options (the top 6 lines in the example), followed by descriptions of the options for the various images. An option in an image description will override a global option.

GLOBAL OPTIONS

There are many possible keywords. The description below is almost literally from user.tex (just slightly abbreviated).

backup=*backup-file*
Copy the original boot sector to *backup-file* (which may also be a device, e.g. */dev/null)* instead of */boot/boot.NNNN*.

boot=*boot-device*
Sets the name of the device (e.g. a hard disk partition) that contains the boot sector. If this keyword is omitted, the boot sector is read from (and possibly written to) the device that is currently mounted as root.

compact
Tries to merge read requests for adjacent sectors into a single read request. This drastically reduces load time and keeps the map smaller. Using `compact' is especially recommended when booting from a floppy disk.

default=*name*
Uses the specified image as the default boot image. If

`default' is omitted, the image appearing first in the configuration file is used.

delay=*tsecs*
Specifies the number of tenths of a second the boot loader should wait before booting the first image. This is useful on systems that immediately boot from the hard disk after enabling the keyboard. The boot loader doesn't wait if `delay' is omitted or is set to zero.

disk=*device-name*
Defines non-standard parameters for the specified disk. See section "Disk geometry" of user.tex for details.

disktab=*disktab-file*
Specifies the name of the disk parameter table. The map installer looks for */etc/disktab* if `disktab' is omitted. The use of disktabs is discouraged.

fix-table
This allows lilo to adjust 3D addresses in partition tables. Each partition entry contains a 3D (sector/head/cylinder) and a linear address of the first and the last sector of the partition. If a partition is not track-aligned and if certain other operating systems (e.g. PC/MS-DOS or OS/2) are using the same disk, they may change the 3D address. lilo can store its boot sector only on partitions where both address types correspond. lilo re-adjusts incorrect 3D start addresses if `fix-table' is set.

WARNING: This does not guarantee that other operating systems may not attempt to reset the address later. It is also possible that this change has other, unexpected side-effects. The correct fix is to re-partition the drive with a program that does align partitions to tracks. Also, with some disks (e.g. some large EIDE disks with address translation enabled), under some circumstances, it may even be unavoidable to have conflicting partition table entries.

force-backup=*backup-file*
Like `backup', but overwrite an old backup copy if it exists.

ignore-table

tells lilo to ignore corrupt partition tables.

install=*boot-sector*
Install the specified file as the new boot sector. If `install' is omitted, */boot/boot.b* is used as the default.

linear
Generate linear sector addresses instead of sector/head/cylinder addresses. Linear addresses are translated at run time and do not depend on disk geometry. Note that boot disks may not be portable if `linear' is used, because the BIOS service to determine the disk geometry does not work reliably for floppy disks. When using `linear' with large disks, */sbin/lilo* may generate references to inaccessible disk areas, because 3D sector addresses are not known before boot time.

lock Enables automatic recording of boot command lines as the defaults for the following boots. This way, lilo "locks" on a choice until it is manually overridden.

map=*map-file*
Specifies the location of the map file. If `map' is omitted, the file */boot/map* is used.

message=*message-file*
specifies a file containing a message that is displayed before the boot prompt. No message is displayed while waiting for a shifting key after printing "LILO ". In the message, the FF character ([Ctrl L]) clears the local screen. The size of the message file is limited to 65535 bytes. The map file has to be rebuilt if the message file is changed or moved.

nowarn
Disables warnings about possible future dangers.

optional
The per-image option `optional' (see below) applies to all images.

password=*password*
The per-image option `password=...' (see below) applies to all images.

prompt
forces entering the boot prompt without expecting any prior key-presses. Unattended reboots are impossible if `prompt' is set and `timeout' isn't.

restricted
The per-image option `restricted' (see below) applies to all images.

serial=*parameters*
enables control from a serial line. The specified serial port is initialized and the boot loader is accepting input from it and from the PC's keyboard. Sending a break on the serial line corresponds to pressing a shift key on the console in order to get the boot loader's attention. All boot images should be password-protected if the serial access is less secure than access to the console, e.g. if the line is connected to a modem. The parameter string has the following syntax:

<port>[,<bps>[<parity>[<bits>]]]

<port>: the number of the serial port, zero-based. 0 corresponds to COM1 alias /dev/ttyS0, etc. All four ports can be used (if present).

<bps>: the baud rate of the serial port. The following baud rates are supported: 110, 150, 300, 600, 1200, 2400, 4800 and 9600 bps. Default is 2400 bps.

<parity>: the parity used on the serial line. The boot loader ignores input parity and strips the 8th bit. The following (upper or lower case) characters are used to describe the parity: n for no parity, e for even parity and o for odd parity.

<bits>: the number of bits in a character. Only 7 and 8 bits are supported. Default is 8 if parity is "none", 7 if parity is "even" or "odd".

If `serial' is set, the value of `delay' is automatically raised to 20.

Example: serial=0,2400n8 initializes COM1 with the

default parameters.

timeout=*tsecs*
sets a timeout (in tenths of a second) for keyboard input. If no key is pressed for the specified time, the first image is automatically booted. Similarly, password input is aborted if the user is idle for too long. The default timeout is infinite.

verbose=*level*
Turns on lots of progress reporting. Higher numbers give more verbose output. If -v is additionally specified on the lilo command line, the level is increased accordingly. The maximum verbosity level is 5.

Additionally, the kernel configuration parameters **append**, **ramdisk**, **read-only**, and **vga** can be set in the global options section. They are used as defaults if they aren't specified in the configuration sections of the respective kernel images.

PER-IMAGE SECTION

A per-image section starts with either a line

image=*pathname*

(to indicate a file or device containing the boot image of a Linux kernel), or a line

other=*pathname*

to indicate an arbitrary system to boot.

In the former case, if an **image** line specifies booting from a device, then one has to indicate the range of sectors to be mapped using

range=*start-end*

In the latter case (booting another system) there are the three options

loader=*chain-loader*
This specifies the chain loader that should be used. By default */boot/chain.b* is used. The chain loader must be specified if booting from a device other than the first hard or floppy disk.

table=*device*
This specifies the device that contains the partition table. The boot loader will not pass partition information to the booted operating system if this variable is omitted. (Some operating systems have other means to determine from which partition they have been booted. E.g., MS-DOS usually stores the geometry of the boot disk or partition in its boot sector.) Note that /sbin/lilo must be re-run if a partition table mapped referenced with `table' is modified.

unsafe
Do not access the boot sector at map creation time. This disables some sanity checks, including a partition table check. If the boot sector is on a fixed-format floppy disk device, using UNSAFE avoids the need to put a readable disk into the drive when running the map installer. `unsafe' and `table' are mutually incompatible.

In both cases the following options apply.

label=*name*
The boot loader uses the main file name (without its path) of each image specification to identify that image. A different name can be used by setting the variable `label'.

alias=*name*
A second name for the same entry can be used by specifying an alias.

lock (See above.)

optional
Omit the image if it is not available at map creation

time. This is useful to specify test kernels that are not always present.

password=*password*
Protect the image by a password.

restricted
A password is only required to boot the image if parameters are specified on the command line (e.g. single).

KERNEL OPTIONS

If the booted image is a Linux kernel, then one may pass command line parameters to this kernel.

append=*string*
Appends the options specified to the parameter line passed to the kernel. This is typically used to specify parameters of hardware that can't be entirely auto-detected or for which probing may be dangerous. Example:

```
append = "hd=64,32,202"
```

literal=*string*
Like `append', but removes all other options (e.g. setting of the root device). Because vital options can be removed unintentionally with `literal', this option cannot be set in the global options section.

ramdisk=*size*
This specifies the size of the optional RAM disk. A value of zero indicates that no RAM disk should be created. If this variable is omitted, the RAM disk size configured into the boot image is used.

read-only
This specifies that the root file system should be mounted read-only. Typically, the system startup procedure re-mounts the root file system read-write later

(e.g. after fsck'ing it).

read-write
This specifies that the root file system should be mounted read-write.

root=*root-device*
This specifies the device that should be mounted as root. If the special name **current** is used, the root device is set to the device on which the root file system is currently mounted. If the root has been changed with -r , the respective device is used. If the variable `root' is omitted, the root device setting contained in the kernel image is used. (And that is set at compile time using the ROOT_DEV variable in the kernel Makefile, and can later be changed with the rdev(8) program.)

vga=*mode*
This specifies the VGA text mode that should be selected when booting. The following values are recognized (case is ignored):
normal: select normal 80x25 text mode.

extended (or **ext**): select 80x50 text mode.

ask: stop and ask for user input (at boot time).

<number>: use the corresponding text mode. A list of available modes can be obtained by booting with vga=ask and pressing [Enter].

If this variable is omitted, the VGA mode setting contained in the kernel image is used. (And that is set at compile time using the SVGA_MODE variable in the kernel Makefile, and can later be changed with the rdev(8) program.)

SEE ALSO

lilo(8), rdev(8).
The lilo distribution comes with very extensive documentation of which the above is an extract.

TOC

NAME

locale - Describes a locale definition file

DESCRIPTION The **locale** definition files contains all the information that the **localedef(1)** command needs to convert it into the binary locale database.

The definition files consist of sections which each describe a locale category in detail.

SYNTAX

The locale definition file starts with a header, that may consist of the following keywords:

<*escape_char*>
: is followed by a character that should be used as the escape-character for the rest of the file to mark characters that should be interpreted in a special way. It defaults to the backslash (\ ).

<*comment_char*>
: is followed by a character that will be used as the comment-character for the rest of the file. It defaults to the number sign (#).

The locale definitions is divided it one part for each locale category. Each part can be copied from another existing locale or can be defined from scratch. If the category should be copied, the only valid keyword in the definition is **copy** followed by the name of the locale which should be copied.

LC_CTYPE

LC_CTYPE category starts with the string *LC_CTYPE* in the first column.

There are the following keywords allowed:

upper
followed by a list of uppercase letters. The letters **A** trough **Z** are included automatically. Characters also specified as **cntrl**, **digit**, **punct**, or **space** are not allowed.

lower
followed by a list of lowercase letters. The letters **a** trough **z** are included automatically. Characters also specified as **cntrl**, **digit**, **punct**, or **space** are not allowed.

alpha
followed by a list of letters. All character specified as either **upper** or **lower** are automatically included. Characters also specified as **cntrl**, **digit**, **punct**, or **space** are not allowed.

digit
followed by the characters classified as numeric digits. Only the digits **0** trough **9** are allowed. They are included by default in this class.

space
followed by a list of characters defined as white-space characters. Characters also specified as **upper**, **lower**, **alpha**, **digit**, **graph**, or **xdigit** are not allowed. The characters **<space>**, **<form-feed>**, **<newline>**, **<carriage-return>**, **<tab>**, and **<vertical-tab>** are automatically included.

cntrl
followed by a list of control characters. Characters also specified as **upper**, **lower**, **alpha**, **digit**, **punct**,

graph, or **xdigit** are not allowed.

punct
followed by a list of punctuation characters. Characters also specified as **upper**, **lower**, **alpha**, **digit**, **cntrl**, **xdigit** or the **<space>** character are not allowed.

graph
followed by a list of printable characters, not including the **<space>** character. The characters defined as **upper**, **lower**, **alpha**, **digit**, **xdigit** and **punct** are automatically included. Characters also specified as **cntrl** are not allowed.

print
followed by a list of printable characters, including the **<space>** character. The characters defined as **upper**, **lower**, **alpha**, **digit**, **xdigit**, **punct** and the **<space>** character are automatically included. Characters also specified as **cntrl** are not allowed.

xdigit
followed by a list of characters classified as hexadecimal digits. The decimal digits must be included followed by one or more set of six characters in ascending order. The following characters are included by default: **0** trough **9**, **a** trough **f**, **A** trough **F**.

blank
followed by a list of characters classified as **blank**. The characters **<space>** and **<tab>** are automatically included.

toupper
followed by a list of mappings from lowercase to uppercase letters. Each mapping is a pair of a lowercase and an uppercase letter separated with a , and enclosed in parentheses. The members of the list are separated with semicolons.

tolower
followed by a list of mappings from uppercase to lower-

case letters. If the keyword tolower is not present, the reverse of the toupper list is used.

The **LC_CTYPE** definition ends with the string *END LC_CYTPE*.

LC_COLLATE

The **LC_COLLATE** category defines the rules for collating characters. Due to limitations of libc not all POSIX-options are implemented.

The definition starts with the string **LC_COLLATE** in the first column.

There are the following keywords allowed:

collating-element

collating-symbol

The order-definition starts with a line:

order_start

followed by a list of keywords out of **forward**, **backward** or **position**. The order definition consists of lines that describe the order and is terminated with the keyword

order_end.

For more details see the sources in **/usr/lib/nls/src** notably the examples **POSIX**, **Example** and **Example2**

The **LC_COLLATE** definition ends with the string *END LC_COLLATE*.

LC_MONETARY

The definition starts with the string **LC_MONETARY** in the first column.

There are the following keywords allowed:

int_curr_symbol
followed by the international currency symbol. This must be a four character string containing the international currency symbol as defined by the ISO 4217 standard (three characters) followed by a separator.

currency_symbol
followed by the local currency symbol.

mon_decimal_point
followed by the string that will be used as the decimal delimiter when formatting monetary quantities.

mon_thousands_sep
followed by the string that will be used as a group separator when formatting monetary quantities.

mon_grouping
followed by a string that describes the formatting of numeric quantities.

positive_sign
followed by a string that is used to indicate a positive sign for monetary quantities.

negative_sign
followed by a string that is used to indicate a negative sign for monetary quantities.

int_frac_digits
followed by the number of fractional digits that should be used when formatting with the **int_curr_symbol.**

frac_digits
followed by the number of fractional digits that should be used when formatting with the **currency_symbol.**

p_cs_precedes
followed by an integer set to **1** if the *currency_symbol* or *int_curr_symbol*
should precede the formatted monetary quantity or set to **0** if the symbol succeeds the value.

p_sep_by_space
followed by an integer.

0 means that no space should be printed between the symbol and the value.

1 means that a space should be printed between the symbol and the value.

2 means that a space should be printed between the symbol and the sign string, if adjacent.

n_cs_precedes

0 - the symbol succeeds the value

1 - the symbol precedes the value

n_sep_by_space
An integer set to **0** if no space separates the *currency_symbol* or *int_curr_symbol* from the value for a negative monetary quantity, set to **1** if a space separates the symbol from the value and set to **2** if a space separates the symbol and the sign string, if adjacent.

p_sign_posn

0 Parentheses enclose the quantity and the *currency_symbol* or *int_curr_symbol*.

1 The sign string precedes the quantity and the *currency_symbol* or the *int_curr_symbol*.

2 The sign string succeeds the quantity and the *currency_symbol* or the *int_curr_symbol*.

3 The sign string precedes the *currency_symbol* or the *int_curr_symbol*.

4 The sign string succeeds the *currency_symbol* or the *int_curr_symbol*.

n_sign_posn

0 Parentheses enclose the quantity and the *currency_symbol* or *int_curr_symbol*.

1 The sign string precedes the quantity and the *currency_symbol* or the *int_curr_symbol*.

2 The sign string succeeds the quantity and the *currency_symbol* or the *int_curr_symbol*.

3 The sign string precedes the *currency_symbol* or the *int_curr_symbol*.

4 The sign string succeeds the *currency_symbol* or the *int_curr_symbol*.

The **LC_MONETARY** definition ends with the string *END LC_MONETARY*.

LC_NUMERIC

The definition starts with the string **LC_NUMERIC** in the first column.

There are the following keywords allowed:

decimal_point
followed by the string that will be used as the decimal delimiter when formatting numeric quantities.

thousands_sep
followed by the string that will be used as a group separator when formatting numeric quantities.

grouping
followed by a string that describes the formatting of numeric quantities.

The **LC_NUMERIC** definition ends with the string *END LC_NUMERIC*.

LC_TIME

The definition starts with the string **LC_TIME** in the first column.

There are the following keywords allowed:

abday
followed by a list of abbreviated weekday names. The list starts with the Sunday or it's translation.

day followed by a list of weekday names. The list starts with the Sunday.

abmon
followed by a list of abbreviated month names.

mon followed by a list of month names.

am_pm
The appropriate representation of the **am** and **pm** strings.

d_t_fmt
The appropriate date and time format.

d_fmt
The appropriate date format.

t_fmt
The appropriate time format.

t_fmt_ampm
The appropriate time format when using 12h clock format.

The **LC_TIME** definition ends with the string *END LC_TIME*.

LC_MESSAGES

The definition starts with the string **LC_MESSAGES** in the first column.

There are the following keywords allowed:

yesexpr
followed by a regular expression that describes possible yes-responses.

noexpr
followed by a regular expression that describes possi-

ble no-responses.

The **LC_MESSAGES** definition ends with the string *END LC_MESSAGES*.

See the POSIX.2 standard for details.

FILES

/usr/lib/locale/ - database for the current locale setting of that category /usr/lib/nls/charmap/* - charmap-files

BUGS

The manpage isn't complete.

AUTHOR

Jochen Hein (Hein@Student.TU-Clausthal.de)

CONFORMING TO

POSIX.2

SEE ALSO

setlocale(3), **localeconv**(3), **charmap**(5), **locale**(1), **localedef**(1)

NAME

motd - message of the day

DESCRIPTION

The contents of **/etc/motd** are displayed by **login**(1) after a successful login but just before it executes the login shell.

The "motd" stands for "message of the day", and this file has been traditionally been used for exactly that (it requires much less disk space than mail to all users).

FILES

/etc/motd

SEE ALSO

login(1) **issue**(5)

TOC

NAME

nologin - prevent non-root users from log into the system

DESCRIPTION

If the file **/etc/nologin** exists, **login**(1) will allow access only to root. Other users will be shown the contents of this file and their logins refused.

FILES

/etc/nologin

SEE ALSO

login(1), **shutdown**(8)


TOC

NAME

nsswitch.conf - System Databases and Name Service Switch configuration file

DESCRIPTION

Various functions in the C Library need to be configured to work correctly in the local environment. Traditionally, this was done by using files (e.g., `/etc/passwd'), but other nameservices (like the Network Information Service (NIS) and the Domain Name Service (DNS)) became popular, and were hacked into the C library, usually with a fixed search order.

The Linux libc5 with NYS support and the GNU C Library 2.x (libc.so.6) contain a cleaner solution of this problem. It is designed after a method used by Sun Microsystems in the C library of Solaris 2. We follow their name and call this scheme "Name Service Switch" (NSS). The sources for the "databases" and their lookup order are specified in the **/etc/nsswitch.conf** file.

The following databases are available in the NSS:

aliases
Mail aliases, used by **sendmail**(8)

ethers
Ethernet numbers

group
Groups of users, used by **getgrent**(3) functions.

hosts
Host names and numbers, used by **gethostbyname**(3) and similar functions.

netgroup
Network wide list of hosts and users, used for access rules

network
Network names and numbers, used by **getnetent**(3) functions.

passwd
User passwords, used by **getpwent**(3) functions.

protocols
Network protocols, used by **getprotoent**(3) functions.

publickey
Public and secret keys for secure_rpc used by NIS+ and NFS.

rpc Remote procedure call names and numbers, used by **getrpcbyname**(3) and similar functions.

services
Network services, used by **getservent**(3) functions.

shadow
Shadow user passwords, used by **getspnam**(3)

An example **/etc/nsswitch.conf** file could be look like (This is also the default if **/etc/nsswitch.conf** is missing):

```
passwd:         compat
group:          compat
shadow:         compat
hosts:          dns [!UNAVAIL=return] files
networks:       nis [NOTFOUND=return] files
ethers:         nis [NOTFOUND=return] files
protocols:      nis [NOTFOUND=return] files
rpc:            nis [NOTFOUND=return] files
services:       nis [NOTFOUND=return] files
```

The first column is the database as you can guess from the table above. The rest of the line specifies how the lookup process works. You can specify the way it works for each database individually.

The configuration specification for each database can con-

tain two different items:
* The service specification like `files', `db', or `nis'.
* The reaction on lookup result like `[NOTFOUND=return]'.

For libc5 with NYS, the allowed service specifications are `files', `nis' and `nisplus'. For hosts, you could specify `dns' as extra service, for passwd and group `compat', but not for shadow.

For GNU C Library, you must have a file called **/lib/libnss_SERVICE.so.1** for every SERVICE you are using. On a standard installation, you could use `files', `db', `nis' and `nisplus'. For hosts, you could specify `dns' as extra service, for passwd, group and shadow `compat'. This Services will not be used by libc5 with NYS.

The second item in the specification gives the user much finer control on the lookup process. Action items are placed between two service names and are written within brackets. The general form is

where
9 STATUS => success | notfound | unavail | tryagain
ACTION => return | continue

The case of the keywords is insignificant. The STATUS values are the results of a call to a lookup function of a specific service. They mean:

success
No error occurred and the wanted entry is returned. The default action for this is `return'.

notfound
The lookup process works ok but the needed value was not found. The default action is `continue'.

unavail
The service is permanently unavailable. This can either mean the needed file is not available, or, for DNS, the server is not available or does not allow queries. The default action is `continue'.

tryagain
The service is temporarily unavailable. This could mean a file is locked or a server currently cannot accept more connections. The default action is `continue'.

Interaction with +/- syntax (compat mode)
Linux libc5 without NYS does not has the name service switch but does allow the user some policy control. In **/etc/passwd** you could have entries of the form +user or +@netgroup (include the specified user from the NIS passwd map), -user or -@netgroup (exclude the specified user) and + (include every user, except the excluded ones, from the NIS passwd map). Since most people only put a + at the end of **/etc/passwd** to include everything from NIS, the switch provides a faster alternative for this case (`passwd: files nis') which doesn't require the single + entry in **/etc/passwd**, **/etc/group** and **/etc/shadow**. If this is not sufficient, the NSS `compat' service provides full +/- semantics. By default, the source is `nis', but this may be overriden by specifying `nisplus' as source for the pseudo-databases **passwd_compat**, **group_compat** and **shadow_compat**. This pseudo-databases are only available in GNU C Library.

FILES

A service named SERVICE is implemented by a shared object library named libnss_SERVICE.so.1 that resides in /lib.

9

	C Library 2.x
/lib/libnss_db.so.1	implements `db' source for GNU C Library 2.x
/lib/libnss_dns.so.1	implements `dns' source for GNU C Library 2.x
/lib/libnss_files.so.1	implements `files' source for GNU C Library 2.x
/lib/libnss_hesoid.so.1	implements `hesoid' source for GNU C Library 2.x
/lib/libnss_nis.so.1	implements `nis' source for GNU C Library 2.x
/lib/libnss_nisplus.so.1	implements `nisplus' source for GNU

NOTES

Within each process that uses nsswitch.conf, the entire file is read only once; if the file is later changed, the process will continue using the old configuration.
With Solaris, is isn't possible to link programs using the NSS Service statically. With Linux, this is no problem.

NAME

passwd - password file

DESCRIPTION

Passwd is a text file, that contains a list of the system's accounts, giving for each account some useful information like user ID, group ID, home directory, shell, etc. Often it also contains the encrypted passwords for each account. It should have general read permission (many utilities, like **ls**(1) use it to map user IDs to user names), but write access only for the superuser.

In the good old days there was no great problem with this general read permission. Everybody could read the encrypted passwords, but the hardware was too slow to crack a well-chosen password, and moreover, the basic assumption used to be that of a friendly user-community. These days many people run some version of the shadow password suite, where */etc/passwd* has *'s instead of encrypted passwords, and the encrypted passwords are in */etc/shadow* which is readable by the superuser only.

Regardless of whether shadow passwords are used, many sysadmins use a star in the encrypted password field to make sure that this user can not authenticate him- or herself using a password. (But see the Notes below.)

If you create a new login, first put a star in the password field, then use **passwd**(1) to set it.

There is one entry per line, and each line has the format:

account:password:UID:GID:GECOS:directory:shell

The field descriptions are:

account the name of the user on the system. It should not contain capital letters.

password the encrypted user password or a star.

UID the numerical user ID.

GID the numerical primary group ID for this user.

GECOS This field is optional and only used for informational purposes. Usually, it contains the full user name. GECOS means General Electric Comprehensive Operating System, which has been renamed to GCOS when GE's large systems division was sold to Honeywell. Dennis Ritchie has reported: "Sometimes we sent printer output or batch jobs to the GCOS machine. The gcos field in the password file was a place to stash the information for the $IDENTcard. Not elegant."

directory the user's $HOME directory.

shell the program to run at login (if empty, use **/bin/sh**). If set to a non-existing executable, the user will be unable to login through **login**(1).

NOTE

If you want to create user groups, their GIDs must be equal and there must be an entry in */etc/group*, or no group will exist.

If the encrypted password is set to a star, the user will be unable to login using **login**(1), but may still login using **rlogin**(1), run existing processes and initiate new ones through **rsh**(1) or **cron**(1) or **at**(1) or mail filters etc. Trying to lock an account by simply changing the shell field yields the same result and additionally allows the use of

su(1).

FILES

/etc/passwd

SEE ALSO

passwd(1), **login**(1), **su**(1), **group**(5), **shadow**(5)

TOC

NAME

proc - process information pseudo-filesystem

DESCRIPTION

/proc is a pseudo-filesystem which is used as an interface to kernel data structures rather than reading and interpreting /dev/kmem. Most of it is read-only, but some files allow kernel variables to be changed.

The following outline gives a quick tour through the /proc hierarchy.

[*number*]
There is a numerical subdirectory for each running process; the subdirectory is named by the process ID. Each contains the following pseudo-files and directories.

cmdline
This holds the complete command line for the process, unless the whole process has been swapped out, or unless the process is a zombie. In either of these later cases, there is nothing in this file: i.e. a read on this file will return as having read 0 characters. This file is null-terminated, but not newline-terminated.

cwd This is a link to the current working directory of the process. To find out the cwd of process 20, for instance, you can do this:
cd /proc/20/cwd; /bin/pwd

Note that the pwd command is often a shell builtin, and might not work properly in this context.

environ
This file contains the environment for the

process. The entries are separated by null characters, and there may be a null character at the end. Thus, to print out the environment of process 1, you would do:
(cat /proc/1/environ; echo) | tr "\000" "\n"

(For a reason why one should want to do this, see *lilo*(8).)

exe a pointer to the binary which was executed, and appears as a symbolic link. *readlink(2)* on the exe special file returns a string in the format:

[device]:inode

For example, [0301]:1502 would be inode 1502 on device major 03 (IDE, MFM, etc. drives) minor 01 (first partition on the first drive).

Also, the symbolic link can be dereferenced normally - attempting to open "exe" will open the executable. You can even type */proc/*[*number*]*/exe* to run another copy of the same process as [number].

find(1) with the -inum option can be used to locate the file.

fd This is a subdirectory containing one entry for each file which the process has open, named by its file descriptor, and which is a symbolic link to the actual file (as the exe entry does). Thus, 0 is standard input, 1 standard output, 2 standard error, etc.

Programs that will take a filename, but will not take the standard input, and which write to a file, but will not send their output to standard output, can be effectively foiled this way, assuming that -i is the flag designating an input file and -o is the flag designating an output file:
foobar -i /proc/self/fd/0 -o /proc/self/fd/1 ...
and you have a working filter. Note that this will not work for programs that seek on their files, as the files in the fd directory are not seekable.

/proc/self/fd/N is approximately the same as /dev/fd/N in some UNIX and UNIX-like systems.

Most Linux MAKEDEV scripts symbolically link /dev/fd to /proc/self/fd, in fact.

maps A file containing the currently mapped memory regions and their access permissions.

The format is:

```
address           perms offset   dev   inode
00000000-0002f000 r-x-- 00000400 03:03 1401
0002f000-00032000 rwx-p 0002f400 03:03 1401
00032000-0005b000 rwx-p 00000000 00:00 0
60000000-60098000 rwx-p 00000400 03:03 215
60098000-600c7000 rwx-p 00000000 00:00 0
bfffa000-c0000000 rwx-p 00000000 00:00 0
```

where address is the address space in the process that it occupies, perms is a set of permissions:

r = read
w = write
x = execute
s = shared
p = private (copy on write)

offset is the offset into the file/whatever, dev is the device (major:minor), and inode is the inode on that device. 0 indicates that no inode is associated with the memory region, as the case would be with bss.

mem This is not the same as the mem (1,1) device, despite the fact that it has the same device numbers. The /dev/mem device is the physical memory before any address translation is done, but the mem file here is the memory of the process that accesses it. This cannot be *mmap(2)*'ed currently, and will not be until a general *mmap(2)* is added to the kernel. (This might have happened by the time you read this.)

mmap Directory of maps by *mmap(2)* which are symbolic links like exe, fd/*, etc. Note that maps includes a superset of this information, so /proc/*/mmap should be considered obsolete.

"0" is usually libc.so.4.

/proc//mmap* was removed in Linux kernel version 1.1.40. (It really **was** obsolete!)

root Unix and linux support the idea of a per-process root of the filesystem, set by the *chroot(2)*

system call. Root points to the file system root, and behaves as exe, fd/*, etc. do.

stat Status information about the process. This is used by *ps(1)*.

The fields, in order, with their proper *scanf(3)* format specifiers, are:

pid %d
The process id.

comm %s
The filename of the executable, in parentheses. This is visible whether or not the executable is swapped out.

state %c
One character from the string "RSDZT" where R is running, S is sleeping in an interruptible wait, D is sleeping in an uninterruptible wait or swapping, Z is zombie, and T is traced or stopped (on a signal).

ppid %d
The PID of the parent.

pgrp %d
The process group ID of the process.

session %d
The session ID of the process.

tty %d
The tty the process uses.

tpgid %d
The process group ID of the process which currently owns the tty that the process is connected to.

flags %u
The flags of the process. Currently, every flag has the math bit set, because crt0.s checks for math emulation, so this is not included in the output. This is probably a bug, as not every process is a compiled C program. The math bit should be a decimal 4, and the traced bit is decimal 10.

minflt %u
The number of minor faults the process has made, those which have not required loading a memory page from disk.

cminflt %u
The number of minor faults that the process and its children have made.

majflt %u
The number of major faults the process has made, those which have required loading a memory page from disk.

cmajflt %u
The number of major faults that the process and its children have made.

utime %d
The number of jiffies that this process has been scheduled in user mode.

stime %d
The number of jiffies that this process has been scheduled in kernel mode.

cutime %d
The number of jiffies that this process and its children have been scheduled in user mode.

cstime %d
The number of jiffies that this process and its children have been scheduled in kernel mode.

counter %d
The current maximum size in jiffies of the process's next timeslice, or what is currently left of its current timeslice, if it is the currently running process.

priority %d
The standard nice value, plus fifteen. The value is never negative in the kernel.

timeout %u
The time in jiffies of the process's next timeout.

itrealvalue %u
The time (in jiffies) before the next SIGALRM is sent to the process due to an interval timer.

system
starttime %d Time the process started in jiffies after boot.

vsize %u
Virtual memory size

rss %u
Resident Set Size: number of pages the process has in real memory, minus 3 for administrative purposes. This is just the pages which count towards text, data, or stack space. This does not include pages which have not been demand-loaded in, or which are swapped out.

rlim %u
Current limit in bytes on the rss of the process (usually 2,147,483,647).

startcode %u
The address above which program text can run.

endcode %u
The address below which program text can run.

startstack %u
The address of the start of the stack.

kstkesp %u
The current value of esp (32-bit stack pointer), as found in the kernel stack page for the process.

kstkeip %u
The current EIP (32-bit instruction pointer).

signal %d
The bitmap of pending signals (usually 0).

blocked %d
The bitmap of blocked signals (usually 0, 2 for shells).

sigignore %d

The bitmap of ignored signals.

sigcatch %d
The bitmap of catched signals.

wchan %u
This is the "channel" in which the process is waiting. This is the address of a system call, and can be looked up in a namelist if you need a textual name. (If you have an up-to-date /etc/psdatabase, then try *ps* -**l** to see the WCHAN field in action)

cpuinfo
This is a collection of CPU and system architecture dependent items, for each supported architecture a different list. The only two common entries are *cpu* which is (guess what) the CPU currently in use and *BogoMIPS* a system constant which is calculated during kernel initialization.

devices
Text listing of major numbers and device groups. This can be used by MAKEDEV scripts for consistency with the kernel.

dma This is a list of the registered *ISA* DMA (direct memory access) channels in use.

filesystems
A text listing of the filesystems which were compiled into the kernel. Incidentally, this is used by *mount(1)* to cycle through different filesystems when none is specified.

interrupts
This is used to record the number of interrupts per each IRQ on (at least) the i386 architechure. Very easy to read formatting, done in ASCII.

ioports
This is a list of currently registered Input-Output port regions that are in use.

kcore
This file represents the physical memory of the system and is stored in the core file format. With this pseudo-file, and an unstripped kernel (/usr/src/linux/tools/zSystem) binary, GDB can be used

to examine the current state of any kernel data structures.

The total length of the file is the size of physical memory (RAM) plus 4KB.

kmsg This file can be used instead of the *syslog(2)* system call to log kernel messages. A process must have superuser privileges to read this file, and only one process should read this file. This file should not be read if a syslog process is running which uses the *syslog(2)* system call facility to log kernel messages.

Information in this file is retrieved with the *dmesg(8)* program).

ksyms
This holds the kernel exported symbol definitions used by the *modules(X)* tools to dynamically link and bind loadable modules.

loadavg
The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes. They are the same as the load average numbers given by *uptime(1)* and other programs.

malloc
This file is only present if CONFIGDEBUGMALLOC was defined during compilation.

meminfo
This is used by *free(1)* to report the amount of free and used memory (both physical and swap) on the system as well as the shared memory and buffers used by the kernel.

It is in the same format as *free(1)*, except in bytes rather than KB.

modules
A text list of the modules that have been loaded by the system.

net various net pseudo-files, all of which give the status of some part of the networking layer. These files contain ASCII structures, and are therefore readable with cat. However, the standard *netstat(8)* suite provides much cleaner access to these files.

arp This holds an ASCII readable dump of the kernel ARP table used for address resolutions. It will show both dynamically learned and pre-programmed ARP entries. The format is:

```
IP address       HW type     Flags       HW address
10.11.100.129    0x1         0x6         00:20:8A:00:0C:5A
10.11.100.5      0x1         0x2         00:C0:EA:00:00:4E
44.131.10.6      0x3         0x2         GW4PTS
```

Where 'IP address' is the IPv4 address of the machine, the 'HW type' is the hardware type of the address from RFC 826. The flags are the internal flags of the ARP structure (as defined in /usr/include/linux/if_arp.h) and the 'HW address' is the physical layer mapping for that IP address if it is known.

dev The dev pseudo-file contains network device status information. This gives the number of received and sent packets, the number of errors and collisions and other basic statistics. These are used by the *ifconfig(8)* program to report device status. The format is:

```
Inter-|   Receive                                                |  Transmit
 face |packets errs drop fifo frame|packets errs drop fifo colls carrier
    lo:       0    0    0    0     0     2353    0    0    0     0    0
  eth0: 644324    1    0    0     1   563770    0    0    0   581    0
```

ipx No information.

ipx_route
No information.

rarp This file uses the same format as the *arp* file and contains the current reverse mapping database used to provide *rarp(8)* reverse address lookup services. If RARP is not configured into the kernel this file will not be present.

raw Holds a dump of the RAW socket table. Much of the information is not of use apart from debugging. The 'sl' value is the kernel hash slot for the socket, the 'local address' is the local address and protocol number pair."St" is the internal status of the socket. The "tx_queue" and "rx_queue" are the outgoing and incoming data queue in terms of kernel memory usage. The "tr", "tm->when" and "rexmits" fields are not used by RAW. The uid field holds the creator euid of the socket.

route
No information, but looks similar to *route(8)*

snmp This file holds the ASCII data needed for the IP, ICMP, TCP and UDP management information bases for an snmp agent. As of writing the TCP mib is incomplete. It is hoped to have it completed by 1.2.0.

tcp Holds a dump of the TCP socket table. Much of the information is not of use apart from debugging. The "sl" value is the kernel hash slot for the socket, the "local address" is the local address and port number pair. The "remote address" is the remote address and port number pair (if connected). 'St' is the internal status of the socket. The 'tx_queue' and 'rx_queue' are the outgoing and incoming data queue in terms of kernel memory usage. The "tr", "tm->when" and "rexmits" fields hold internal information of the kernel socket state and are only useful for debugging. The uid field holds the creator euid of the socket.

udp Holds a dump of the UDP socket table. Much of the information is not of use apart from debugging. The "sl" value is the kernel hash slot for the socket, the "local address" is the local address and port number pair. The "remote address" is the remote address and port number pair (if connected). "St" is the internal status of the socket. The "tx_queue" and "rx_queue" are the outgoing and incoming data queue in terms of kernel memory usage. The "tr", "tm->when" and "rexmits" fields are not used by UDP. The uid field holds the creator euid of the socket. The format is:

```
sl  local_address rem_address   st tx_queue rx_queue tr rexmits  tm->when uid
 1: 01642C89:0201 0C642C89:03FF 01 00000000:00000001 01:000071BA 00000000 0
 1: 00000000:0801 00000000:0000 0A 00000000:00000000 00:00000000 6F000100 0
 1: 00000000:0201 00000000:0000 0A 00000000:00000000 00:00000000 00000000 0
```

unix Lists the UNIX domain sockets present within the system and their status. The format is:

```
9
     Num RefCount Protocol Flags    Type St Path
     0: 00000002 00000000 00000000 0001 03
     1: 00000001 00000000 00010000 0001 01 /dev/printer
9
```

Where 'Num' is the kernel table slot number, 'RefCount' is the number of users of the socket, 'Protocol' is

currently always 0, 'Flags' represent the internal kernel flags holding the status of the socket. Type is always '1' currently (Unix domain datagram sockets are not yet supported in the kernel). 'St' is the internal state of the socket and Path is the bound path (if any) of the socket.

pci This is a listing of all PCI devices found during kernel initialization and their configuration.

scsi A directory with the scsi midlevel pseudo-file and various SCSI lowlevel driver directories, which contain a file for each SCSI host in this system, all of which give the status of some part of the SCSI IO subsystem. These files contain ASCII structures, and are therefore readable with cat.

You can also write to some of the files to reconfigure the subsystem or switch certain features on or off.

scsi This is a listing of all SCSI devices known to the kernel. The listing is similar to the one seen during bootup. scsi currently supports only the *singledevice* command which allows root to add a hotplugged device to the list of known devices.

An **echo 'scsi singledevice 1 0 5** will cause host scsi1 to scan on SCSI channel 0 for a device on ID 5 LUN 0. If there is already a device known on this address or the address is invalid an error

drivername

drivername can currently be: NCR53c7xx, aha152x, aha1542, aha1740, aic7xxx, buslogic, eata_dma, eata_pio, fdomain, in2000, pas16, qlogic, scsi_debug, seagate, t128, u15-24f, ultrastore or wd7000. These directories show up for all drivers which registered at least one SCSI HBA. Every directory contains one file per registered host. Every host-file is named after the number the host got assigned during initilization.

Reading these files will usually show driver and host configuration, statistics etc.

Writing to these files allows different things on different hosts. For example with the *latency* and *nolatency* commands root can switch on and off command latency measurement code in the eata_dma

driver. With the *lockup* and *unlock* commands root can control bus lockups simulated by the scsi_debug driver.

self This directory refers to the process accessing the /proc filesystem, and is identical to the /proc directory named by the process ID of the same process.

stat kernel/system statistics

cpu *3357* **0** *4313 1362393*
The number of jiffies (1/100ths of a second) that the system spent in user mode, user mode with low priority (nice), system mode, and the idle task, respectively. The last value should be 100 times the second entry in the uptime pseudo-file.

disk **0 0 0 0**
The four disk entries are not implemented at this time. I'm not even sure what this should be, since kernel statistics on other machines usually track both transfer rate and I/Os per second and this only allows for one field per drive.

page 5741 1808
The number of pages the system paged in and the number that were paged out (from disk).

swap **1 0**
The number of swap pages that have been brought in and out.

The number of interrupts received from the system boot.

ctxt 115315
The number of context switches that the system underwent.

btime 769041601
boot time, in seconds since the epoch (January 1, 1970).

sys This directory (present since 1.3.57) contains a number of files and subdirectories corresponding to kernel variables. These variables can be read and sometimes modified using the *proc* file system, and using the *sysctl*(2) system call. Presently, there are subdirectories *kernel*, *net*, *vm* that each contain more

files and subdirectories.

kernel
This contains files *domainname*, *file-max*, *file-nr*, *inode-max*, *inode-nr*, *osrelease*, *panic*, *real-root-dev*, *securelevel*, with function fairly clear from the name.

The (read-only) file *file-nr* gives the number of files presently opened.

The file *file-max* gives the maximum number of open files the kernel is willing to handle. If 1024 is not enough for you, try
echo 4096 > /proc/sys/kernel/file-max

Similarly, the files *inode-nr* and *inode-max* indicate the present and the maximum number of inodes.

The files *ostype*, *osrelease*, *version* give substrings of */proc/version*.

The file *panic* gives r/w access to the kernel variable *panic_timeout*. If this is zero, the kernel will loop on a panic; if nonzero it indicates that the kernel should autoreboot after this number of seconds.

The file *securelevel* seems rather meaningless at present - root is just too powerful.

uptime
This file contains two numbers: the uptime of the system (seconds), and the amount of time spent in idle process (seconds).

This strings identifies the kernel version that is currently running. For instance:
Linux version 1.0.9 (quinlan@phaze) #1 Sat May 14 01:51:54 EDT 1994

SEE ALSO

cat(1), find(1), free(1), mount(1), ps(1), tr(1), uptime(1), readlink(2), mmap(2), chroot(2), syslog(2), hier(7), arp(8), dmesg(8), netstat(8), route(8), ifconfig(8), procinfo(8) and much more

CONFORMS TO

This roughly conforms to a Linux 1.3.11 kernel. Please update this as necessary!

Last updated for Linux 1.3.11.

CAVEATS

Note that many strings (i.e., the environment and command line) are in the internal format, with sub-fields terminated by NUL bytes, so you may find that things are more readable if you use *od* **-c** or *tr* "*\000*" "**\n**" to read them.

This manual page is incomplete, possibly inaccurate, and is the kind of thing that needs to be updated very often.

BUGS

The */proc* file system may introduce security holes into processes running with **chroot**(2). For example, if */proc* is mounted in the **chroot** hierarchy, a **chdir**(2) to */proc/***1***/root* will return to the original root of the file system. This may be considered a feature instead of a bug, since Linux does not yet support the **fchroot**(2) call.

TOC

TOC

NAME

protocols - the protocols definition file

DESCRIPTION

This file is a plain ASCII file, describing the various DARPA internet protocols that are available from the TCP/IP subsystem. It should be consulted instead of using the numbers in the ARPA include files, or, even worse, just guessing them. These numbers will occur in the protocol field of any ip header.

Keep this file untouched since changes would result in incorrect ip packages. Protocol numbers and names are specified by the DDN Network Information Center.

Each line is of the following format:

protocol number aliases ...

where the fields are delimited by spaces or tabs. Empty lines and lines starting with a hash mark (#) are ignored. Remainder of lines are also ignored from the occurrence of a hash mark.

The field descriptions are:

protocol
the native name for the protocol. For example ip, tcp or udp.

number
the official number for this protocol as it will appear within the ip header.

aliases
optional aliases for the protocol.

This file might be distributed over a network using a network-workwide naming service like Yellow Pages/NIS or BIND/Hesoid.

FILES

/etc/protocols
The protocols definition file.

SEE ALSO

getprotoent(3)

Guide to Yellow Pages Service

Guide to BIND/Hesiod Service

TOC

TOC

NAME

securetty - file which lists ttys from which root can log in

DESCRIPTION

/etc/securetty is used by **login**(1); the file contains the device names of tty lines (one per line, without leading */dev/*) on which root is allowed to login.

FILES

/etc/securetty

SEE ALSO

login(1)

TOC

NAME

services - Internet network services list

DESCRIPTION

services is a plain ASCII file providing a mapping between friendly textual names for internet services, and their underlying assigned port numbers and protocol types. Every networking program should look into this file to get the port number (and protocol) for its service. The C library routines **getservent**(3), **getservbyname**(3), **getservbyport**(3), **setservent**(3), and **endservent**(3) support querying this file from programs.

Port numbers are assigned by the IANA (Internet Assigned Numbers Authority), and their current policy is to assign both TCP and UDP protocols when assigning a port number. Therefore, most entries will have two entries, even for TCP only services.

Port numbers below 1024 (so-called 'low numbered' ports) can only be bound to by root (see **bind**(2), **tcp**(7), and **udp**(7).) This is so that clients connecting to low numbered ports can trust that the service running on the port is the standard implementation, and not a rogue service run by a user of the machine. Well-known port numbers specified by the IANA are normally located in this root only space.

The presence of an entry for a service in the **services** file does not necessarily mean that the service is currently running on the machine. See **inetd.conf**(5) for the configuration of Internet services offered. Note that not all networking services are started by **inetd**(8), and so won't appear in **inetd.conf**(5). In particular, news (NNTP) and mail (SMTP) servers are often initialised from the system boot scripts.

The location of the **services** file is defined by **_PATH_SERVICES** in */usr/include/netdb.h*. This is usually set to */etc/services*.

Each line describes one service, and is of the form:

service-name *port/protocol* [*aliases* ...]

where:

service-name
is the friendly name the service is known by and looked up under. It is case sensitive. Often, the client program is named after the *service-name*.

port is the port number (in decimal) to use for this service.

protocol is the type of protocol to be used. This field should match an entry in the **protocols**(5) file. Typical values include **tcp** and **udp**.

aliases is an optional space or tab separated list of other names for this service (but see the BUGS section below). Again, the names are case sensitive.

Either spaces or tabs may be used to separate the fields.

Comments are started by the hash sign (#) and continue until the end of the line. Blank lines are skipped.

The *service-name* should begin in the first column of the file, since leading spaces are not stripped. *service-names* can be any printable characters excluding space and tab, however, a conservative choice of characters should be used to minimise inter-operability problems. Eg: a-z, 0-9, and hyphen (-) would seem a sensible choice.

Lines not matching this format should not be present in the file. (Currently, they are silently skipped by **getservent**(3), **getservbyname**(3), and **getservbyport**(3). However, this behaviour should not be relied on.)

As a backwards compatibility feature, the slash (/) between the *port* number and *protocol* name can in fact be either a

slash or a comma (,). Use of the comma in modern installations is depreciated.

This file might be distributed over a network using a network-wide naming service like Yellow Pages/NIS or BIND/Hesiod.

A sample **services** file might look like this:

```
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp          # message send protocol
msp             18/udp          # message send protocol
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp             21/tcp
# 22 - unassigned
telnet          23/tcp
```

BUGS

There is a maximum of 35 aliases, due to the way the **getservent**(3) code is written.

Lines longer than **BUFSIZ** (currently 1024) characters will be ignored by **getservent**(3), **getservbyname**(3), and **getservbyport**(3). However, this will also cause the next line to be mis-parsed.

FILES

/etc/services
The Internet network services list

/usr/include/netdb.h
Definition of **_PATH_SERVICES**

TOC

SEE ALSO

getservent(3), **getservbyname**(3), **getservbyport**(3), **setservent**(3), **endservent**(3), **protocols**(5), **listen**(2), **inetd.conf**(5), **inetd**(8).

Assigned Numbers RFC, most recently RFC 1700, (AKA STD0002)

Guide to Yellow Pages Service

Guide to BIND/Hesiod Service

NAME

shells - pathnames of valid login shells

DESCRIPTION

/etc/shells is a text file which contains the full pathnames of valid login shells. This file is consulted by **chsh**(1) and available to be queried by other programs.

EXAMPLES

/etc/shells may contain the following paths:

/bin/sh
/bin/csh

FILES

/etc/shells

SEE ALSO

chsh(1)

TOC

NAME

termcap - terminal capability database

DESCRIPTION

The termcap database is an obsolete facility for describing the capabilities of character-cell terminals and printers. It is retained only for capability with old programs; new ones should use the **terminfo**(5) database and associated libraries.

/etc/termcap is an ASCII file (the database master) that lists the capabilities of many different types of terminal. Programs can read termcap to find the particular escape codes needed to control the visual attributes of the terminal actually in use. (Other aspects of the terminal are handled by stty.) The termcap database is indexed on the TERM environment variable.

Termcap entries must be defined on a single logical line, with `\' used to suppress the newline. Fields are separated by `:'. The first field of each entry starts at the left-hand margin, and contains a list of names for the terminal, separated by '|'.

The first subfield may (in BSD termcap entries from versions 4.3 and prior) contain a short name consisting of two characters. This short name may consist of capital or small letters. In 4.4BSD termcap entries this field is omitted.

The second subfield (first, in the newer 4.4BSD format) contains the name used by the environment variable TERM. It should be spelled in lowercase letters. Selectable hardware capabilities should be marked by appending a hyphen and a suffix to this name. See below for an example. Usual suffixes are w (more than 80 characters wide), am (automatic margins), nam (no automatic margins) and rv (reverse video display). The third subfield contains a long and descriptive name for this termcap entry.

Subsequent fields contain the terminal capabilities; any continued capability lines must be indented one tab from the left margin.

Although there is no defined order, it is suggested to write first boolean, then numeric and at last string capabilities, each sorted alphabetically without looking at lower or upper spelling. Capabilities of similar functions can be written in one line.

Example for:

```
Head line: vt|vt101|DEC VT 101 terminal in 80 character mode:\
Head line: Vt|vt101-w|DEC VT 101 terminal in (wide) 132 character mode:\
Boolean: :bs:\
Numeric: :co#80:\
String: :sr=\E[H:\
```

Boolean Capabilities

5i	Printer will not echo on screen
am	Automatic margins which means automatic line wrap
bs	Control-H (8 dec.) performs a backspace
bw	Backspace on left margin wraps to previous line and right margin
da	Display retained above screen
db	Display retained below screen
eo	A space erases all characters at cursor position
es	Escape sequences and special characters work in status line
gn	Generic device
hc	This is a hardcopy terminal
HC	The cursor is hard to see when not on bottom line
hs	Has a status line
hz	Hazeltine bug, the terminal can not print tilde characters
in	Terminal inserts nulls, not spaces, to fill whitespace
km	Terminal has a meta key
mi	Cursor movement works in insert mode
ms	Cursor movement works in standout/underline mode
NP	No pad character
NR	ti does not reverse te
nx	No padding, must use XON/XOFF
os	Terminal can overstrike
ul	Terminal underlines although it can not overstrike
xb	Beehive glitch, f1 sends ESCAPE, f2 sends ^C
xn	Newline/wraparound glitch
xo	Terminal uses xon/xoff protocol
xs	Text typed over standout text will be displayed in standout
xt	Teleray glitch, destructive tabs and odd standout mode

Numeric Capabilities

co	Number of columns
dB	Delay in milliseconds for backspace on hardcopy terminals
dC	Delay in milliseconds for carriage return on hardcopy terminals
dF	Delay in milliseconds for form feed on hardcopy terminals
dN	Delay in milliseconds for new line on hardcopy terminals
dT	Delay in milliseconds for tabulator stop on hardcopy terminals
dV	Delay in milliseconds for vertical tabulator stop on hardcopy terminals

it	Difference between tab positions
lh	Height of soft labels
lm	Lines of memory
lw	Width of soft labels
li	Number of lines
Nl	Number of soft labels
pb	Lowest baud rate which needs padding
sg	Standout glitch
ug	Underline glitch
vt	virtual terminal number
ws	Width of status line if different from screen width

String Capabilities

!1	shifted save key
!2	shifted suspend key
!3	shifted undo key
#1	shifted help key
#2	shifted home key
#3	shifted input key
#4	shifted cursor left key
%0	redo key
%1	help key
%2	mark key
%3	message key
%4	move key
%5	next-object key
%6	open key
%7	options key
%8	previous-object key
%9	print key
%a	shifted message key
%b	shifted move key
%c	shifted next key
%d	shifted options key
%e	shifted previous key
%f	shifted print key
%g	shifted redo key
%h	shifted replace key
%i	shifted cusor right key
%j	shifted resume key
&0	shifted cancel key
&1	reference key
&2	refresh key
&3	replace key
&4	restart key
&5	resume key
&6	save key
&7	suspend key
&8	undo key
&9	shifted begin key
*0	shifted find key
*1	shifted command key


```
*2   shifted copy key
*3   shifted create key
*4   shifted delete character
*5   shifted delete line
*6   select key
*7   shifted end key
*8   shifted clear line key
*9   shifted exit key
@0   find key
@1   begin key
@2   cancel key
@3   close key
@4   command key
@5   copy key
@6   create key
@7   end key
@8   enter/send key
@9   exit key
al   Insert one line
AL   Indert %1 lines
ac   Pairs of block grafic characters to map alternate character set
ae   End alternative character set
as   Start alternative character set for block grafic characters
bc   Backspace, if not ^H
bl   Audio bell
bt   Move to previous tab stop
cb   Clear from beginning of line to cursor
cc   Dummy command character
cd   Clear to end of screen
ce   Clear to end of line
ch   Move cursor horizontally only to column %1
cl   Clear screen and cursor home
cm   Cursor move to row %1 and column %2 (on screen)
CM   Move cursor to row %1 and column %2 (in memory)
cr   Carriage return
cs   Scroll region from line %1 to %2
ct   Clear tabs
cv   Move cursor vertically only to line %1
dc   Delete one character
DC   Delete %1 characters
dl   Delete one line
DL   Delete %1 lines
dm   Begin delete mode
do   Cursor down one line
DO   Cursor down #1 lines
ds   Disable status line
eA   Enable alternate character set
ec   Erase %1 characters starting at cursor
ed   End delete mode
ei   End insert mode
ff   Formfeed character on hardcopy terminals
fs   Return character to its position before going to status line
```


```
F1   The string sent by function key f11
F2   The string sent by function key f12
F3   The string sent by function key f13
...  ...
F9   The string sent by function key f19
FA   The string sent by function key f20
FB   The string sent by function key f21
...  ...
FZ   The string sent by function key f45
Fa   The string sent by function key f46
Fb   The string sent by function key f47
...  ...
Fr   The string sent by function key f63
hd   Move cursor a half line down
ho   Cursor home
hu   Move cursor a half line up
i1   Initialization string 1 at login
i3   Initialization string 3 at login
is   Initialization string 2 at login
ic   Insert one character
IC   Insert %1 characters
if   Initialization file
im   Begin insert mode
ip   Insert pad time and needed special characters after insert
iP   Initialization program
K1   upper left key on keypad
K2   center key on keypad
K3   upper right key on keypad
K4   bottom left key on keypad
K5   bottom right key on keypad
k0   Function key 0
k1   Function key 1
k2   Function key 2
k3   Function key 3
k4   Function key 4
k5   Function key 5
k6   Function key 6
k7   Function key 7
k8   Function key 8
k9   Function key 9
k;   Function key 10
ka   Clear all tabs key
kA   Insert line key
kb   Backspace key
kB   Back tab stop
kC   Clear screen key
kd   Cursor down key
kD   Key for delete character under cursor
ke   turn keypad off
kE   Key for clear to end of line
kF   Key for scolling forward/down
kh   Cursor home key
```


```
kH   Cursor hown down key
kI   Insert character/Insert mode key
kl   Cursor left key
kL   Key for delete line
kM   Key for exit insert mode
kN   Key for next page
kP   Key for previous page
kr   Cursor right key
kR   Key for scolling backward/up
ks   Turn keypad on
kS   Clear to end of screen key
kt   Clear this tab key
kT   Set tab here key
ku   Cursor up key
l0   Label of zeroth function key, if not f0
l1   Label of first function key, if not f1
l2   Label of first function key, if not f2
...  ...
la   Label of tenth function key, if not f10
le   Cursor left one character
ll   Move cursor to lower left corner
LE   Cursor left %1 characters
LF   Turn soft labels off
LO   Turn soft labels on
mb   Start blinking
MC   Clear soft margins
md   Start bold mode
me   End all mode like so, us, mb, md and mr
mh   Start half bright mode
mk   Dark mode (Characters invisible)
ML   Set left soft margin
mm   Put terminal in meta mode
mo   Put terminal out of meta mode
mp   Turn on protected attribute
mr   Start reverse mode
MR   Set right soft margin
nd   Cursor right one character
nw   Carriage return command
pc   Padding character
pf   Turn printer off
pk   Program key %1 to send string %2 as if typed by user
pl   Program key %1 to execute string %2 in local mode
pn   Program soft label %1 to to show string %2
po   Turn the printer on
pO   Turn the printer on for %1 (<256) bytes
ps   Print screen contents on printer
px   Program key %1 to send string %2 to computer
r1   Reset string 1 to set terminal to sane modes
r2   Reset string 2 to set terminal to sane modes
r3   Reset string 3 to set terminal to sane modes
RA   disable automatic margins
rc   Restore saved cursor position
```


```
rf   Reset string file name
RF   Request for input from terminal
RI   Cursor right %1 characters
rp   Repeat character %1 for %2 times
rP   Padding after character sent in replace mode
rs   Reset string
RX   Turn off XON/XOFF flow control
sa   Set %1 %2 %3 %4 %5 %6 %7 %8 %9 attributes
SA   enable automatic margins
sc   Save cursor position
se   End standout mode
sf   Normal scroll one line
SF   Normal scroll %1 lines
so   Start standout mode
sr   Reverse scroll
SR   scroll back %1 lines
st   Set tabulator stop in all rows at current column
SX   Turn on XON/XOFF flow control
ta   move to next hardware tab
tc   Read in terminal description from another entry
te   End program that uses cursor motion
ti   Begin program that uses cursor motion
ts   Move cursor to column %1 of status line
uc   Underline character under cursor and move cursor right
ue   End underlining
up   Cursor up one line
UP   Cursor up %1 lines
us   Start underlining
vb   Visible bell
ve   Normal cursor visible
vi   Cursor unvisible
vs   Standout cursor
wi   Set window from line %1 to %2 and column %3 to %4
XF   XOFF character if not ^S

There are several ways of defining  the  control  codes  for
string capabilities:

Normal Characters except '^','\' and '%' repesent themself.

A '^x' means Control-x.  Control-A equals 1 decimal.

\x means a special code.  x can  be  one  of  the  following
charaters:
     E Escape (27)
     n Linefeed (10)
     r Carriage return (13)
     t Tabulation (9)
     b Backspace (8)
     f Form feed (12)
     0 Null character.  A \xxx specifies the octal character
     xxx.
```

i Increments paramters by one.

r Single parameter capability

+ Add value of next character to this parameter and do binary output

2 Do ASCII output of this parameter with a field with of 2

d Do ASCII output of this parameter with a field with of 3

% Print a '%'

If you use binary output, then you should avoid the null character because it terminates the string. You should reset tabulator expansion if a tabulator can be the binary output of a parameter.

Warning:
The above metacharacters for parameters may be wrong, they document Minix termcap which may not be compatible with Linux termcap.

The block grafic characters can be specified by three string capabilities:

as start the alternative charset

ae end it

ac pairs of characters. The first character is the name of the block grafic symbol and the second characters is its definition.

The following names are available:

+ right arrow (>)
, left arrow (<)
. down arrow (v)
0 full square (#)
I latern (#)
- upper arrow (^)
' rhombus (+)
a chess board (:)
f degree (')
g plus-minus (#)
h square (#)
j right bottom corner (+)
k right upper corner (+)


```
l    left upper corner (+)
m    left bottom corner (+)
n    cross (+)
o    upper horizontal line (-)
q    middle horizontal line (-)
s    bottom horizontal line (_)
t    left tee (+)
u    right tee (+)
v    bottom tee (+)
w    normal tee (+)
x    vertical line (|)
~    paragraph (???)

The values in parentheses are suggested defaults  which  are
used by curses, if the capabilities are missing.
```

SEE ALSO

termcap(3), **curses**(3), **terminfo**(5)

TOC

TOC

NAME

ttytype - terminal device to default terminal type mapping

DESCRIPTION

The */etc/ttytype* file associates termcap/terminfo terminal type names with tty lines. Each line consists of a terminal type, followed by whitespace, followed by a tty name (a device name without the */dev/*) prefix.

This association is used by the program **tset**(1) to set the environment variable TERM to the default terminal name for the user's current tty.

This facility was designed for a traditional time-sharing environment featuring character-cell terminals hardwired to a Unix minicomputer. It is little used on modern workstation and personal Unixes.

EXAMPLE

A typical */etc/ttytype* is:

```
con80x25 tty1
vt320 ttys0
```

FILES

/etc/ttytype
the tty definitions file.

SEE ALSO

getty(1), **terminfo**(5), **termcap**(5)

NAME

utmp, wtmp - login records

SYNOPSIS

```
#include <utmp.h>
```

DESCRIPTION

The **utmp** file allows one to discover information about who is currently using the system. There may be more users currently using the system, because not all programs use utmp logging.

Warning: utmp must not be writable, because many system programs (foolishly) depend on its integrity. You risk faked system logfiles and modifications of system files if you leave **utmp** writable to any user.

The file is a sequence of entries with the following structure declared in the include file (note that this is only one of several definitions around; details depend on the version of libc):

```
    #define UT_UNKNOWN          0
    #define RUN_LVL             1
    #define BOOT_TIME           2
    #define NEW_TIME            3
    #define OLD_TIME            4
    #define INIT_PROCESS        5
    #define LOGIN_PROCESS       6
    #define USER_PROCESS        7
    #define DEAD_PROCESS        8
    #define ACCOUNTING          9

    #define UT_LINESIZE         12
```


```
#define UT_NAMESIZE           32
#define UT_HOSTSIZE           256

struct exit_status {
  short int e_termination;    /* process termination status.  */
  short int e_exit;           /* process exit status.  */
};

struct utmp {
  short ut_type;              /* type of login */
  pid_t ut_pid;               /* pid of login process */
  char ut_line[UT_LINESIZE];  /* device name of tty - "/dev/" */
  char ut_id[4];              /* init id or abbrev. ttyname */
  char ut_user[UT_NAMESIZE];  /* user name */
  char ut_host[UT_HOSTSIZE];  /* hostname for remote login */
  struct exit_status ut_exit; /* The exit status of a process
                                 marked as DEAD_PROCESS. */
  long ut_session;            /* session ID, used for windowing*/
  struct timeval ut_tv;       /* time entry was made.  */
  int32_t ut_addr_v6[4];      /* IP address of remote host.  */
  char pad[20];               /* Reserved for future use.  */
};

/* Backwards compatibility hacks.  */
#define ut_name ut_user
#ifndef _NO_UT_TIME
#define ut_time ut_tv.tv_sec
#endif
#define ut_xtime ut_tv.tv_sec
#define ut_addr ut_addr_v6[0]
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of login in the form of *time*(2). String fields are terminated by '**\0**' if they are shorter than the size of the field.

The first entries ever created result from *init*(8) processing *inittab*(5). Before an entry is processed, though, *init*(8) cleans up utmp by setting **ut_type** to **DEAD_PROCESS**, clearing **ut_user**, **ut_host** and **ut_time** with null bytes for each record which **ut_type** is not **DEAD_PROCESS** or **RUN_LVL** and where no process with PID **ut_pid** exists. If no empty record with the needed **ut_id** can be found, init creates a new one. It sets **ut_id** from the inittab, **ut_pid** and **ut_time** to the current values and **ut_type** to **INIT_PROCESS**.

getty(8) locates the entry by the pid, changes **ut_type** to


TOC

LOGIN_PROCESS, changes **ut_time**, sets **ut_line** and waits for connection to be established. *login*(8), after a user has been authenticated, changes **ut_type** to **USER_PROCESS**, changes **ut_time** and sets **ut_host** and **ut_addr**. Depending on *getty*(8) and *login*(8), records may be located by **ut_line** instead of the preferable **ut_pid**.

When *init*(8) finds that a process has exited, it locates its utmp entry by **ut_pid**, sets **ut_type** to **DEAD_PROCESS** and clears **ut_user**, **ut_host** and **ut_time** with null bytes.

xterm(1) and other terminal emulators directly create a **USER_PROCESS** record and generate the **ut_id** by using the last two letters of **/dev/ttyp%c** or by using **p%d** for **/dev/pts/%d**. If they find a **DEAD_PROCESS** for this id, they recycle it, otherwise they create a new entry. If they can, they will mark it as **DEAD_PROCESS** on exiting and it is advised that they null **ut_line**, **ut_time**, **ut_user** and **ut_host** as well.

xdm(8) should not create an utmp record, because there is no assigned terminal. Letting it create one will result in trouble like: finger: can not stat /dev/machine.dom. It should create wtmp entries, though, just like *ftpd*(8) does.

telnetd(8) sets up a **LOGIN_PROCESS** entry and leaves the rest to *login*(8) as usual. After the telnet session ends, *telnetd*(8) cleans up utmp in the described way.

The **wtmp** file records all logins and logouts. Its format is exactly like **utmp** except that a null user name indicates a logout on the associated terminal. Furthermore, the terminal name **"~"** with user name **"shutdown"** or **"reboot"** indicates a system shutdown or reboot and the pair of terminal names **"|"/"}"** logs the old/new system time when *date*(1) changes it. **wtmp** is maintained by *login*(1), and *init*(1) and some versions of *getty*(1). Neither of these programs creates the file, so if it is removed record-keeping is turned off.

FILES

/var/run/utmp
/var/log/wtmp

TOC

CONFORMING TO

Linux utmp entries conform neither to v7/BSD nor to SYSV: They are a mix of the two. v7/BSD has fewer fields; most importantly it lacks **ut_type**, which causes native v7/BSD-like programs to display (for example) dead or login entries. Further there is no configuration file which allocates slots to sessions. BSD does so, because it lacks **ut_id** fields. In Linux (as in SYSV), the **ut_id** field of a record will never change once it has been set, which reserves that slot without needing a configuration file. Clearing **ut_id** may result in race conditions leading to corrupted utmp entries and and potential security holes. Clearing the above mentioned fields by filling them with null bytes is not required by SYSV semantics, but it allows to run many programs which assume BSD semantics and which do not modify utmp. Linux uses the BSD conventions for line contents, as documented above.

SYSV only uses the type field to mark them and logs informative messages such as e.g. **"new time"** in the line field. **UT_UNKNOWN** seems to be a Linux invention. SYSV has no **ut_host** or **ut_addr_v6** fields.

Unlike various other systems, where utmp logging can be disabled by removing the file, utmp must always exist on Linux. If you want to disable *who*(1) then do not make utmp world readable.

Note that the utmp struct from libc5 has changed in libc6. Because of this, binaries using the old libc5 struct will corrupt */var/run/utmp* and/or */var/log/wtmp*. Debian systems include a patched libc5 which uses the new utmp format. The problem still exists with wtmp since it's accessed directly in libc5.

RESTRICTIONS

The file format is machine dependent, so it is recommended that it be processed only on the machine architecture where it got created.

BUGS

This manpage is based on the libc5 one, things may work differently now.

SEE ALSO

ac(1), **date**(1), **getutent**(3), **init**(8), **last**(1), **login**(1), **updwtmp**(3), **who**(1)

TOC

Linux Man Pages Section 6

- intro.6

NAME

intro - Introduction to games

DESCRIPTION

This chapter describes all the games and funny little programs available on the system.

AUTHORS

Look at the header of the manual page for the author(s) and copyright conditions. Note that these can be different from page to page!


TOC

Linux Man Pages Section 7

- ascii.7
- bootparam.7
- glob.7
- hier.7
- intro.7
- iso_8859_1.7
- latin1.7
- locale.7
- mailaddr.7
- man.7
- regex.7
- signal.7
- suffixes.7
- unicode.7
- utf-8.7

NAME

ascii - the ASCII character set encoded in octal, decimal, and hexadecimal

DESCRIPTION

ASCII is the American Standard Code for Information Interchange. It is a 7-bit code. Many 8-bit codes (such as ISO 8859-1, the Linux default character set) contain ASCII as their lower half. The international counterpart of ASCII is known as ISO 646.

The following table contains the 128 ASCII characters.

C program '\X' escapes are noted.

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0'	100	64	40	@	140	96	60	`
001	1	01	SOH	101	65	41	A	141	97	61	a
002	2	02	STX	102	66	42	B	142	98	62	b
003	3	03	ETX	103	67	43	C	143	99	63	c
004	4	04	EOT	104	68	44	D	144	100	64	d
005	5	05	ENQ	105	69	45	E	145	101	65	e
006	6	06	ACK	106	70	46	F	146	102	66	f
007	7	07	BEL '\a'	107	71	47	G	147	103	67	g
010	8	08	BS '\b'	110	72	48	H	150	104	68	h
011	9	09	HT '\t'	111	73	49	I	151	105	69	i
012	10	0A	LF '\n'	112	74	4A	J	152	106	6A	j
013	11	0B	VT '\v'	113	75	4B	K	153	107	6B	k
014	12	0C	FF '\f'	114	76	4C	L	154	108	6C	l
015	13	0D	CR '\r'	115	77	4D	M	155	109	6D	m
016	14	0E	SO	116	78	4E	N	156	110	6E	n
017	15	0F	SI	117	79	4F	O	157	111	6F	o
020	16	10	DLE	120	80	50	P	160	112	70	p
021	17	11	DC1	121	81	51	Q	161	113	71	q

022	18	12	DC2		122	82	52	R				
023	19	13	DC3		123	83	53	S				
024	20	14	DC4		124	84	54	T				
025	21	15	NAK		125	85	55	U				
026	22	16	SYN		126	86	56	V				
027	23	17	ETB		127	87	57	W				
030	24	18	CAN		130	88	58	X				
031	25	19	EM		131	89	59	Y				
032	26	1A	SUB		132	90	5A	Z				
033	27	1B	ESC		133	91	5B	[				
034	28	1C	FS		134	92	5C	\				'\\'
035	29	1D	GS		135	93	5D	]				
036	30	1E	RS		136	94	5E	^				
037	31	1F	US		137	95	5F	_				
040	32	20	SPACE		140	96	60	`				
041	33	21	!		141	97	61	a				
042	34	22	"		142	98	62	b				
043	35	23	#		143	99	63	c				
044	36	24	$		144	100	64	d				
045	37	25	%		145	101	65	e				
046	38	26	&		146	102	66	f				
047	39	27	'		147	103	67	g				
050	40	28	(		150	104	68	h				
051	41	29	)		151	105	69	i				
052	42	2A	*		152	106	6A	j				
053	43	2B	+		153	107	6B	k				
054	44	2C	,		154	108	6C	l				
055	45	2D	-		155	109	6D	m				
056	46	2E	.		156	110	6E	n				
057	47	2F	/		157	111	6F	o				
060	48	30	0		160	112	70	p				
061	49	31	1		161	113	71	q				
062	50	32	2		162	114	72	r				
063	51	33	3		163	115	73	s				
064	52	34	4		164	116	74	t				
065	53	35	5		165	117	75	u				
066	54	36	6		166	118	76	v				
067	55	37	7		167	119	77	w				
070	56	38	8		170	120	78	x				
071	57	39	9		171	121	79	y				
072	58	3A	:		172	122	7A	z				
073	59	3B	;		173	123	7B	{				
074	60	3C	<		174	124	7C	|	075	61	3D	=
	175	125	7D	}								
076	62	3E	>		176	126	7E	~				
077	63	3F	?		177	127	7F	DEL				

HISTORY

An **ascii** manual page appeared in Version 7 AT&T UNIX.

On older terminals, the underscore code is displayed as a left arrow, called backarrow, the caret is displayed as an up-arrow and the vertical bar has a hole in the middle.

The ASCII standard was published by the United States of America Standards Institute (USASI) in 1968.

SEE ALSO

iso_8859_1(7)

NAME

bootparam - Introduction to boot time parameters of the Linux kernel

DESCRIPTION

The Linux kernel accepts certain `command line options' or `boot time parameters' at the moment it is started. In general this is used to supply the kernel with information about hardware parameters that the kernel would not be able to determine on its own, or to avoid/override the values that the kernel would otherwise detect.

When the kernel is booted directly by the BIOS (say from a floppy to which you copied a kernel using `cp zImage /dev/fd0'), you have no opportunity to specify any parameters. So, in order to take advantage of this possibility you have to use software that is able to pass parameters, like LILO or loadlin. For a few parameters one can also modify the kernel image itself, using rdev, see **rdev**(8) for further details.

The LILO program (LInux LOader) written by Werner Almesberger is the most commonly used. It has the ability to boot various kernels, and stores the configuration information in a plain text file. (See **lilo**(8) and **lilo.conf**(5).) LILO can boot DOS, OS/2, Linux, FreeBSD, UnixWare, etc., and is quite flexible.

The other commonly used Linux loader is `LoadLin' which is a DOS program that has the capability to launch a Linux kernel from the DOS prompt (with boot-args) assuming that certain resources are available. This is good for people that want to launch Linux from DOS.

It is also very useful if you have certain hardware which

relies on the supplied DOS driver to put the hardware into a known state. A common example is `SoundBlaster Compatible' sound cards that require the DOS driver to twiddle a few mystical registers to put the card into a SB compatible mode. Booting DOS with the supplied driver, and then loading Linux from the DOS prompt with loadlin avoids the reset of the card that happens if one rebooted instead.

THE ARGUMENT LIST

The kernel command line is parsed into a list of strings (boot arguments) separated by spaces. Most of the boot args take the form of:

```
name[=value_1][,value_2]...[,value_10]
```

where `name' is a unique keyword that is used to identify what part of the kernel the associated values (if any) are to be given to. Note the limit of 10 is real, as the present code only handles 10 comma separated parameters per keyword. (However, you can re-use the same keyword with up to an additional 10 parameters in unusually complicated situations, assuming the setup function supports it.)

Most of the sorting goes on in linux/init/main.c. First, the kernel checks to see if the argument is any of the special arguments `root=', `nfsroot=', `nfsaddrs=', `ro', `rw', `debug' or `init'. The meaning of these special arguments is described below.

Then it walks a list of setup functions (contained in the bootsetups array) to see if the specified argument string (such as `foo') has been associated with a setup function (`foo_setup()') for a particular device or part of the kernel. If you passed the kernel the line foo=3,4,5,6 then the kernel would search the bootsetups array to see if `foo' was registered. If it was, then it would call the setup function associated with `foo' (foo_setup()) and hand it the arguments 3, 4, 5 and 6 as given on the kernel command line.

Anything of the form `foo=bar' that is not accepted as a setup funtion as described above is then interpreted as an environment variable to be set. A (useless?) example would be to use `TERM=vt100' as a boot argument.

Any remaining arguments that were not picked up by the kernel and were not interpreted as environment variables are then passed onto process one, which is usually the init program. The most common argument that is passed to the init process is the word `single' which instructs init to boot the computer in single user mode, and not launch all the usual daemons. Check the manual page for the version of init installed on your system to see what arguments it accepts.

GENERAL NON-DEVICE SPECIFIC BOOT ARGS

`init=...'
This sets the initial command to be executed by the kernel. If this is not set, or cannot be found, the kernel will try */etc/init*, then */bin/init*, then */sbin/init*, then */bin/sh* and panic if all of this fails.

`nfsaddrs=...'
This sets the nfs boot address to the given string. This boot address is used in case of a net boot.

`nfsroot=...'
This sets the nfs root name to the given string. If this string does not begin with '/' or ',' or a digit, then it is prefixed by `/tftpboot/'. This root name is used in case of a net boot.

`no387'
(Only when CONFIG_BUGi386 is defined.) Some i387 coprocessor chips have bugs that show up when used in 32 bit protected mode. For example, some of the early ULSI-387 chips would cause solid lockups while performing floating point calculations. Using the `no387' boot arg causes Linux to

ignore the maths coprocessor even if you have one. Of course you must then have your kernel compiled with math emulation support!

`no-hlt'

(Only when CONFIG_BUGi386 is defined.) Some of the early i486DX-100 chips have a problem with the `hlt' instruction, in that they can't reliably return to operating mode after this instruction is used. Using the `no-hlt' instruction tells Linux to just run an infinite loop when there is nothing else to do, and to not halt the CPU. This allows people with these broken chips to use Linux.

`root=...'

This argument tells the kernel what device is to be used as the root filesystem while booting. The default of this setting is determined at compile time, and usually is the value of the root device of the system that the kernel was built on. To override this value, and select the second floppy drive as the root device, one would use `root=/dev/fd1'. (The root device can also be set using **rdev**(8).)

The root device can be specified symbolically or numerically. A symbolic specification has the form /dev/XXYN, where XX designates the device type (`hd' for ST-506 compatible hard disk, with Y in `a'-`d'; `sd' for SCSI compatible disk, with Y in `a'-`e'; `ad' for Atari ACSI disk, with Y in `a'-`e', `ez' for a Syquest EZ135 parallel port removable drive, with Y=`a', `xd' for XT compatible disk, with Y either `a' or `b'; `fd' for floppy disk, with Y the floppy drive number - fd0 would be the DOS `A:' drive, and fd1 would be `B:'), Y the driver letter or number, and N the number (in decimal) of the partition on this device (absent in the case of floppies). Recent kernels allow many other types, mostly for CD-ROMs: nfs, ram, scd, mcd, cdu535, aztcd, cm206cd, gscd, sbpcd, sonycd, bpcd. (The type nfs specifies a net boot; ram refers to a ram disk.)
Note that this has nothing to do with the designation of these devices on your file system. The `/dev/' part is purely conventional.

The more awkward and less portable numeric specification of the above possible root devices in major/minor format is also accepted. (E.g., /dev/sda3 is major 8, minor 3, so you

could use `root=0x803' as an alternative.)

`ro' and `rw'

The `ro' option tells the kernel to mount the root filesystem as `readonly' so that filesystem consistency check programs (fsck) can do their work on a quiescent file system. No processes can write to files on the filesystem in question until it is `remounted' as read/write capable, e.g., by `mount -w -n -o remount /'. (See also **mount**(8).)

The `rw' option tells the kernel to mount the root filesystem read/write. This is the default.

The choice between read-only and read/write can also be set using **rdev**(8).

`reserve=...'

This is used to protect I/O port regions from probes. The form of the command is:

reserve=*iobase*,*extent*[,*iobase*,*extent*]...

In some machines it may be necessary to prevent device drivers from checking for devices (auto-probing) in a specific region. This may be because of hardware that reacts badly to the probing, or hardware that would be mistakenly identified, or merely hardware you don't want the kernel to initialize.

The reserve boot-time argument specifies an I/O port region that shouldn't be probed. A device driver will not probe a reserved region, unless another boot argument explicitly specifies that it do so.

For example, the boot line

reserve=0x300,32 blah=0x300

keeps all device drivers except the driver for `blah' from probing 0x300-0x31f.

`mem=...'

The BIOS call defined in the PC specification that returns the amount of installed memory was only designed to be able to report up to 64MB. Linux uses this BIOS call at boot to determine how much memory is installed. If you have more than 64MB of RAM installed, you can use this boot arg to tell Linux how much memory you have. The value is in decimal or hexadecimal (prefix 0x), and the suffixes `k' (times 1024) or `M' (times 1048576) can be used. Here is a quote from Linus on usage of the `mem=' parameter.

``The kernel will accept any `mem=xx' parameter you give it, and if it turns out that you lied to it, it will crash horribly sooner or later. The parameter indicates the highest addressable RAM address, so `mem=0x1000000' means you have 16MB of memory, for example. For a 96MB machine this would be `mem=0x6000000'.

NOTE NOTE NOTE: some machines might use the top of memory for BIOS cacheing or whatever, so you might not actually have up to the full 96MB addressable. The reverse is also true: some chipsets will map the physical memory that is covered by the BIOS area into the area just past the top of memory, so the top-of-mem might actually be 96MB + 384kB for example. If you tell linux that it has more memory than it actually does have, bad things will happen: maybe not at once, but surely eventually.''

`panic=N'

By default the kernel will not reboot after a panic, but this option will cause a kernel reboot after N seconds (if N > 0). This panic timeout can also be set by "echo N > /proc/sys/kernel/panic".

`reboot=[warm|cold][,[bios|hard]]'

(Only when CONFIG_BUGi386 is defined.) Since 2.0.22 a reboot is by default a cold reboot. One asks for the old default with `reboot=warm'. (A cold reboot may be required to reset certain hardware, but might destroy not yet written data in a disk cache. A warm reboot may be faster.) By default a reboot is hard, by asking the keyboard controller to pulse the reset line low, but there is at least one type of motherboard where that doesn't work. The option `reboot=bios' will instead jump through the BIOS.

`nosmp' and `maxcpus=N'
(Only when __SMP__ is defined.) A command-line option of `nosmp' or `maxcpus=0' will disable SMP activation entirely; an option `maxcpus=N' limits the maximum number of CPUs activated in SMP mode to N.

BOOT ARGUMENTS FOR USE BY KERNEL DEVELOPERS

`debug'
Kernel messages are handed off to the kernel log daemon klogd so that they may be logged to disk. Messages with a priority above *console_loglevel* are also printed on the console. (For these levels, see <linux/kernel.h>.) By default this variable is set to log anything more important than debug messages. This boot argument will cause the kernel to also print the messages of DEBUG priority. The console loglevel can also be set at run time via an option to klogd. See **klogd**(8).

`profile=N'
It is possible to enable a kernel profiling function, if one wishes to find out where the kernel is spending its CPU cycles. Profiling is enabled by setting the variable *prof_shift* to a nonzero value. This is done either by specifying CONFIG_PROFILE at compile time, or by giving the `profile=' option. Now the value that *prof_shift* gets will be N, when given, or CONFIG_PROFILE_SHIFT, when that is given, or 2, the default. The significance of this variable is that it gives the granularity of the profiling: each clock tick, if the system was executing kernel code, a counter is incremented:

```
profile[address >> prof_shift]++;
```

The raw profiling information can be read from */proc/profile*. Probably you'll want to use a tool such as

readprofile.c to digest it. Writing to */proc/profile* will clear the counters.

`swap=N1,N2,N3,N4,N5,N6,N7,N8'
Set the eight parameters max_page_age, page_advance, page_decline, page_initial_age, age_cluster_fract, age_cluster_min, pageout_weight, bufferout_weight that control the kernel swap algorithm. For kernel tuners only.

`buff=N1,N2,N3,N4,N5,N6'
Set the six parameters max_buff_age, buff_advance, buff_decline, buff_initial_age, bufferout_weight, buffermem_grace that control kernel buffer memory management. For kernel tuners only.

BOOT ARGUMENTS FOR RAMDISK USE

(Only if the kernel was compiled with CONFIG_BLK_DEV_RAM.) In general it is a bad idea to use a ramdisk under Linux - the system will use available memory more efficiently itself. But while booting (or while constructing boot floppies) it is often useful to load the floppy contents into a ramdisk. One might also have a system in which first some modules (for filesystem or hardware) must be loaded before the main disk can be accessed.

In Linux 1.3.48, ramdisk handling was changed drastically. Earlier, the memory was allocated statically, and there was a `ramdisk=N' parameter to tell its size. (This could also be set in the kernel image at compile time, or by use of **rdev**(8).) These days ram disks use the buffer cache, and grow dynamically. For a lot of information (e.g., how to use **rdev**(8) in conjunction with the new ramdisk setup), see */usr/src/linux/Documentation/ramdisk.txt*.

There are four parameters, two boolean and two integral.

`load_ramdisk=N'

If N=1, do load a ramdisk. If N=0, do not load a ramdisk. (This is the default.)

`prompt_ramdisk=N'
If N=1, do prompt for insertion of the floppy. (This is the default.) If N=0, do not prompt. (Thus, this parameter is never needed.)

`ramdisk_size=N' or (obsolete) `ramdisk=N'
Set the maximal size of the ramdisk(s) to N kB. The default is 4096 (4 MB).

`ramdisk_start=N'
Sets the starting block number (the offset on the floppy where the ramdisk starts) to N. This is needed in case the ramdisk follows a kernel image.

`noinitrd'
(Only if the kernel was compiled with CONFIG_BLK_DEV_RAM and CONFIG_BLK_DEV_INITRD.) These days it is possible to compile the kernel to use initrd. When this feature is enabled, the boot process will load the kernel and an initial ramdisk; then the kernel converts initrd into a "normal" ramdisk, which is mounted read-write as root device; then /linuxrc is executed; afterwards the "real" root file system is mounted, and the initrd filesystem is moved over to /initrd; finally the usual boot sequence (e.g. invocation of /sbin/init) is performed.

For a detailed description of the initrd feature, see */usr/src/linux/Documentation/initrd.txt*.

The `noinitrd' option tells the kernel that although it was compiled for operation with initrd, it should not go through the above steps, but leave the initrd data under */dev/initrd*. (This device can be used only once - the data is freed as soon as the last process that used it has closed */dev/initrd*.)

BOOT ARGUMENTS FOR SCSI DEVICES

General notation for this section:

iobase -- the first I/O port that the SCSI host occupies. These are specified in hexidecimal notation, and usually lie in the range from 0x200 to 0x3ff.

irq -- the hardware interrupt that the card is configured to use. Valid values will be dependent on the card in question, but will usually be 5, 7, 9, 10, 11, 12, and 15. The other values are usually used for common peripherals like IDE hard disks, floppies, serial ports, etc.

scsi-id -- the ID that the host adapter uses to identify itself on the SCSI bus. Only some host adapters allow you to change this value, as most have it permanently specified internally. The usual default value is 7, but the Seagate and Future Domain TMC-950 boards use 6.

parity -- whether the SCSI host adapter expects the attached devices to supply a parity value with all information exchanges. Specifying a one indicates parity checking is enabled, and a zero disables parity checking. Again, not all adapters will support selection of parity behaviour as a boot argument.

`max_scsi_luns=...'

A SCSI device can have a number of `sub-devices' contained within itself. The most common example is one of the new SCSI CD-ROMs that handle more than one disk at a time. Each CD is addressed as a `Logical Unit Number' (LUN) of that particular device. But most devices, such as hard disks, tape drives and such are only one device, and will be assigned to LUN zero.

Some poorly designed SCSI devices cannot handle being probed for LUNs not equal to zero. Therefore, if the compile time flag CONFIG_SCSI_MULTI_LUN is not set, newer kernels will by default only probe LUN zero.

To specify the number of probed LUNs at boot, one enters `max_scsi_luns=n' as a boot arg, where n is a number between

one and eight. To avoid problems as described above, one would use n=1 to avoid upsetting such broken devices.

SCSI tape configuration

Some boot time configuration of the SCSI tape driver can be achieved by using the following:

st=*buf_size*[,*write_threshold*[,*max_bufs*]]

The first two numbers are specified in units of kB. The default *buf_size* is 32kB, and the maximum size that can be specified is a ridiculous 16384kB. The *write_threshold* is the value at which the buffer is committed to tape, with a default value of 30kB. The maximum number of buffers varies with the number of drives detected, and has a default of two. An example usage would be:

st=32,30,2

Full details can be found in the README.st file that is in the scsi directory of the kernel source tree.

Adaptec aha151x, aha152x, aic6260, aic6360, SB16-SCSI configuration

The aha numbers refer to cards and the aic numbers refer to the actual SCSI chip on these type of cards, including the Soundblaster-16 SCSI.

The probe code for these SCSI hosts looks for an installed BIOS, and if none is present, the probe will not find your card. Then you will have to use a boot arg of the form:

aha152x=*iobase*[,*irq*[,*scsi-id*[,*reconnect*[,*parity*]]]]

If the driver was compiled with debugging enabled, a sixth value can be specified to set the debug level.

All the parameters are as described at the top of this section, and the *reconnect* value will allow device disconnect/reconnect if a non-zero value is used. An example usage is as follows:

aha152x=0x340,11,7,1

Note that the parameters must be specified in order, meaning that if you want to specify a parity setting, then you will have to specify an iobase, irq, scsi-id and reconnect value as well.

Adaptec aha154x configuration

The aha1542 series cards have an i82077 floppy controller onboard, while the aha1540 series cards do not. These are busmastering cards, and have parameters to set the ``fairness'' that is used to share the bus with other devices. The boot arg looks like the following.

aha1542=*iobase*[,*buson*,*busoff*[,*dmaspeed*]]

Valid iobase values are usually one of: 0x130, 0x134, 0x230, 0x234, 0x330, 0x334. Clone cards may permit other values.

The *buson*, *busoff* values refer to the number of microseconds that the card dominates the ISA bus. The defaults are 11us on, and 4us off, so that other cards (such as an ISA LANCE Ethernet card) have a chance to get access to the ISA bus.

The *dmaspeed* value refers to the rate (in MB/s) at which the DMA (Direct Memory Access) transfers proceed. The default is 5MB/s. Newer revision cards allow you to select this value as part of the soft-configuration, older cards use jumpers. You can use values up to 10MB/s assuming that your motherboard is capable of handling it. Experiment with caution if using values over 5MB/s.

Adaptec aha274x, aha284x, aic7xxx configuration

These boards can accept an argument of the form:

aic7xxx=*extended*,*no_reset*

The *extended* value, if non-zero, indicates that extended translation for large disks is enabled. The *no_reset* value, if non-zero, tells the driver not to reset the SCSI bus when setting up the host adaptor at boot.

AdvanSys SCSI Hosts configuration (`advansys=')

The AdvanSys driver can accept up to four i/o addresses that will be probed for an AdvanSys SCSI card. Note that these

values (if used) do not effect EISA or PCI probing in any way. They are only used for probing ISA and VLB cards. In addition, if the driver has been compiled with debugging enabled, the level of debugging output can be set by adding an 0xdeb[0-f] parameter. The 0-f allows setting the level of the debugging messages to any of 16 levels of verbosity.

AM53C974

AM53C974=*host-scsi-id*,*target-scsi-id*,*max-rate*,*max-offset*

BusLogic SCSI Hosts configuration (`BusLogic=')

BusLogic=*N1*,*N2*,*N3*,*N4*,*N5*,*S1*,*S2*,...

For an extensive discussion of the BusLogic command line parameters, see */usr/src/linux/drivers/scsi/BusLogic.c* (lines 3149-3270 in the kernel version I am looking at). The text below is a very much abbreviated extract.

The parameters N1-N5 are integers. The parameters S1,... are strings. N1 is the I/O Address at which the Host Adapter is located. N2 is the Tagged Queue Depth to use for Target Devices that support Tagged Queuing. N3 is the Bus Settle Time in seconds. This is the amount of time to wait between a Host Adapter Hard Reset which initiates a SCSI Bus Reset and issuing any SCSI Commands. N4 is the Local Options (for one Host Adapter). N5 is the Global Options (for all Host Adapters).

The string options are used to provide control over Tagged Queuing (TQ:Default, TQ:Enable, TQ:Disable, TQ:<Per-Target-Spec>), over Error Recovery (ER:Default, ER:HardReset, ER:BusDeviceReset, ER:None, ER:<Per-Target-Spec>), and over Host Adapter Probing (NoProbe, NoProbeISA, NoSortPCI).

EATA/DMA configuration

The default list of i/o ports to be probed can be changed by

eata=*iobase*,*iobase*,....

Future Domain TMC-16x0 configuration

fdomain=*iobase*,*irq*[,*adapter_id*]

Great Valley Products (GVP) SCSI controller configuration

gvp11=*dma_transfer_bitmask*

Future Domain TMC-8xx, TMC-950 configuration

tmc8xx=*mem_base*,*irq*

The *mem_base* value is the value of the memory mapped I/O region that the card uses. This will usually be one of the following values: 0xc8000, 0xca000, 0xcc000, 0xce000, 0xdc000, 0xde000.

IN2000 configuration

in2000=S

where S is a comma-separated string of items keyword[:value]. Recognized keywords (possibly with value) are: ioport:addr, noreset, nosync:x, period:ns, disconnect:x, debug:x, proc:x. For the function of these parameters, see */usr/src/linux/drivers/scsi/in2000.c*.

NCR5380 and NCR53C400 configuration

The boot arg is of the form

ncr5380=*iobase*,*irq*,*dma*

or

ncr53c400=*iobase*,*irq*

If the card doesn't use interrupts, then an IRQ value of 255 (0xff) will disable interrupts. An IRQ value of 254 means to autoprobe. More details can be found in the file */usr/src/linux/drivers/scsi/README.g_NCR5380*.

NCR53C8xx configuration

ncr53c8xx=S

where S is a comma-separated string of items keyword:value. Recognized keywords are: mpar (master_parity), spar (scsi_parity), disc (disconnection), specf (special_features), ultra (ultra_scsi), fsn (force_sync_nego), tags (default_tags), sync (default_sync), verb (verbose), debug (debug), burst (burst_max). For the function of the assigned values, see */usr/src/linux/drivers/scsi/ncr53c8xx.c*.

NCR53c406a configuration

ncr53c406a=*iobase*[,*irq*[,*fastpio*]]

Specify irq = 0 for non-interrupt driven mode. Set fastpio = 1 for fast pio mode, 0 for slow mode.

IOMEGA PPA3 configuration

ppa=*iobase*[,*speed_high*[,*speed_low*[,*nybble*]]]

Here iobase is the parallel port address (default 0x378), speed_high is the port delay in data phase in microseconds (default 1), speed_low is the port delay (in microseconds) otherwise (default 6), and nybble is a boolean `force nybble (4-bit) mode' (default 0=false). See also */usr/src/linux/drivers/scsi/README.ppa*.

Pro Audio Spectrum configuration

The PAS16 uses a NC5380 SCSI chip, and newer models support jumperless configuration. The boot arg is of the form:

pas16=*iobase*,*irq*

The only difference is that you can specify an IRQ value of 255, which will tell the driver to work without using interrupts, albeit at a performance loss. The iobase is usually 0x388.

Seagate ST-0x configuration

If your card is not detected at boot time, you will then have to use a boot arg of the form:

st0x=*mem_base*,*irq*

The *mem_base* value is the value of the memory mapped I/O region that the card uses. This will usually be one of the following values: 0xc8000, 0xca000, 0xcc000, 0xce000, 0xdc000, 0xde000.

Trantor T128 configuration

These cards are also based on the NCR5380 chip, and accept the following options:

t128=*mem_base*,*irq*

The valid values for *mem_base* are as follows: 0xcc000, 0xc8000, 0xdc000, 0xd8000.

UltraStor 14F/34F configuration

The default list of i/o ports to be probed can be changed by

eata=*iobase*,*iobase*,....

WD7000 configuration

wd7000=*irq*,*dma*,*iobase*

Commodore Amiga A2091/590 SCSI controller configuration

wd33c93=S

where S is a comma-separated string of options. Recognized options are nosync:bitmask, nodma:x, period:ns, disconnect:x, debug:x, clock:x, next. For details, see */usr/src/linux/drivers/scsi/wd33c93.c*.

HARD DISKS

IDE Disk/CD-ROM Driver Parameters

The IDE driver accepts a number of parameters, which range from disk geometry specifications, to support for broken controller chips. Drive specific options are specified by using `hdX=' with X in `a'-`h'.

Non-drive specific options are specified with the prefix `hd='. Note that using a drive specific prefix for a non-drive specific option will still work, and the option will just be applied as expected.

Also note that `hd=' can be used to refer to the next unspecified drive in the (a, ..., h) sequence. For the following discussions, the `hd=' option will be cited for brevity. See the file README.ide in linux/drivers/block for more

details.

The `hd=cyls,heads,sects[,wpcom[,irq]]' options

These options are used to specify the physical geometry of the disk. Only the first three values are required. The cylinder/head/sectors values will be those used by fdisk. The write precompensation value is ignored for IDE disks. The IRQ value specified will be the IRQ used for the interface that the drive resides on, and is not really a drive specific parameter.

The `hd=serialize' option

The dual IDE interface CMD-640 chip is broken as designed such that when drives on the secondary interface are used at the same time as drives on the primary interface, it will corrupt your data. Using this option tells the driver to make sure that both interfaces are never used at the same time.

The `hd=dtc2278' option

This option tells the driver that you have a DTC-2278D IDE interface. The driver then tries to do DTC specific operations to enable the second interface and to enable faster transfer modes.

The `hd=noprobe' option

Do not probe for this drive. For example,

```
hdb=noprobe hdb=1166,7,17
```

would disable the probe, but still specify the drive geometry so that it would be registered as a valid block device, and hence useable.

The `hd=nowerr' option

Some drives apparently have the WRERR_STAT bit stuck on permanently. This enables a work-around for these broken devices.

The `hd=cdrom' option

This tells the IDE driver that there is an ATAPI compatible CD-ROM attached in place of a normal IDE hard disk. In most cases the CD-ROM is identified automatically, but if it isn't then this may help.

Standard ST-506 Disk Driver Options (`hd=')

The standard disk driver can accept geometry arguments for the disks similar to the IDE driver. Note however that it only expects three values (C/H/S) -- any more or any less and it will silently ignore you. Also, it only accepts `hd=' as an argument, i.e. `hda=' and so on are not valid here. The format is as follows:

```
hd=cyls,heads,sects
```

If there are two disks installed, the above is repeated with the geometry parameters of the second disk.

XT Disk Driver Options (`xd=')

If you are unfortunate enough to be using one of these old 8 bit cards that move data at a whopping 125kB/s then here is the scoop. If the card is not recognised, you will have to use a boot arg of the form:

```
xd=type,irq,iobase,dma_chan
```

The type value specifies the particular manufacturer of the card, and are as follows: 0=generic; 1=DTC; 2,3,4=Western Digital, 5,6,7=Seagate; 8=OMTI. The only difference between multiple types from the same manufacturer is the BIOS string used for detection, which is not used if the type is specified.

The xd_setup() function does no checking on the values, and assumes that you entered all four values. Don't disappoint it. Here is an example usage for a WD1002 controller with the BIOS disabled/removed, using the `default' XT controller parameters:

```
xd=2,5,0x320,3
```

Syquest's EZ* removable disks

ez=*iobase*[,*irq*[,*rep*[,*nybble*]]]

IBM MCA BUS DEVICES

See also */usr/src/linux/Documentation/mca.txt*.

PS/2 ESDI hard disks

It is possible to specify the desired geometry at boot time:

ed=*cyls*,*heads*,*sectors*.

For a ThinkPad-720, add the option

tp720=1.

IBM Microchannel SCSI Subsystem configuration

ibmmcascsi=N

where N is the *pun* (SCSI ID) of the subsystem.

CD-ROMs (Non-SCSI/ATAPI/IDE)

The Aztech Interface

The syntax for this type of card is:

aztcd=iobase[,magic_number]

If you set the magic_number to 0x79 then the driver will try and run anyway in the event of an unknown firmware version. All other values are ignored.

The MicroSolutions `backpack' CDrom

Syntax:

bpcd=iobase

The CDU-31A and CDU-33A Sony Interface

This CD-ROM interface is found on some of the Pro Audio Spectrum sound cards, and other Sony supplied interface cards. The syntax is as follows:

```
cdu31a=iobase,[irq[,is_pas_card]]
```

Specifying an IRQ value of zero tells the driver that hardware interrupts aren't supported (as on some PAS cards). If your card supports interrupts, you should use them as it cuts down on the CPU usage of the driver.

The *is_pas_card* should be entered as `PAS' if using a Pro Audio Spectrum card, and otherwise it should not be specified at all.

The CDU-535 Sony Interface

The syntax for this CD-ROM interface is:

```
sonycd535=iobase[,irq]
```

A zero can be used for the I/O base as a `placeholder' if one wishes to specify an IRQ value.

The GoldStar Interface

The syntax for this CD-ROM interface is:

```
gscd=iobase
```

The ISP16 CD-ROM Interface

Syntax:

```
isp16=[iobase[,irq[,dma[,type]]]]
```

(three integers and a string). If the type is given as `noisp16', the interface will not be configured. Other recognized types are: `Sanyo", `Sony', `Panasonic' and `Mitsumi'.

The Mitsumi Standard Interface

The syntax for this CD-ROM interface is:

```
mcd=iobase,[irq[,wait_value]]
```

The *wait_value* is used as an internal timeout value for people who are having problems with their drive, and may or may not be implemented depending on a compile time #define. The Mitsumi FX400 is an IDE/ATAPI CD-ROM player and does not use the mcd driver.

The Mitsumi XA/MultiSession Interface

This is for the same hardware as above, but the driver has extended features. Syntax:

```
mcdx=iobase[,irq]
```

The Optics Storage Interface

The syntax for this type of card is:

```
optcd=iobase
```

The Phillips CM206 Interface

The syntax for this type of card is:

```
cm206=[iobase][,irq]
```

The driver assumes numbers between 3 and 11 are IRQ values, and numbers between 0x300 and 0x370 are I/O ports, so you can specify one, or both numbers, in any order. It also accepts `cm206=auto' to enable autoprobing.

The Sanyo Interface

The syntax for this type of card is:

```
sjcd=iobase[,irq[,dma_channel]]
```

The SoundBlaster Pro Interface

The syntax for this type of card is:

```
sbpcd=iobase,type
```

where type is one of the following (case sensitive) strings: `SoundBlaster', `LaserMate', or `SPEA'. The I/O base is that of the CD-ROM interface, and not that of the sound portion of the card.

ETHERNET DEVICES

Different drivers make use of different parameters, but they all at least share having an IRQ, an I/O port base value, and a name. In its most generic form, it looks something like this:

```
ether=irq,iobase[,param_1[,...param_8]],name
```

The first non-numeric argument is taken as the name. The param_n values (if applicable) usually have different meanings for each different card/driver. Typical param_n values are used to specify things like shared memory address, interface selection, DMA channel and the like.

The most common use of this parameter is to force probing for a second ethercard, as the default is to only probe for one. This can be accomplished with a simple:

```
ether=0,0,eth1
```

Note that the values of zero for the IRQ and I/O base in the above example tell the driver(s) to autoprobe.

The Ethernet-HowTo has extensive documentation on using multiple cards and on the card/driver specific implementation of the param_n values where used. Interested readers should refer to the section in that document on their particular card.

THE FLOPPY DISK DRIVER

There are many floppy driver options, and they are all listed in README.fd in linux/drivers/block. This information is taken directly from that file.

floppy=mask,allowed_drive_mask
Sets the bitmask of allowed drives to mask. By default, only units 0 and 1 of each floppy controller are allowed. This is done because certain non-standard hardware (ASUS PCI motherboards) mess up the keyboard when accessing units 2 or 3. This option is somewhat obsoleted by the cmos option.

floppy=all_drives
Sets the bitmask of allowed drives to all drives. Use this if you have more than two drives connected to a floppy controller.

floppy=asus_pci
Sets the bitmask to allow only units 0 and 1. (The default)

floppy=daring
Tells the floppy driver that you have a well behaved floppy controller. This allows more efficient and smoother operation, but may fail on certain controllers. This may speed up certain operations.

floppy=0,daring
Tells the floppy driver that your floppy controller should be used with caution.

floppy=one_fdc
Tells the floppy driver that you have only floppy controller (default)

floppy=two_fdc or floppy=address,two_fdc
Tells the floppy driver that you have two floppy controllers. The second floppy controller is assumed to be at address. If address is not given, 0x370 is assumed.

floppy=thinkpad
Tells the floppy driver that you have a Thinkpad. Thinkpads use an inverted convention for the disk change line.

floppy=0,thinkpad
Tells the floppy driver that you don't have a Thinkpad.

floppy=drive,type,cmos
Sets the cmos type of drive to type. Additionally, this drive is allowed in the bitmask. This is useful if you have more than two floppy drives (only two can be described in the physical cmos), or if your BIOS uses non-standard CMOS types. Setting the CMOS to 0 for the first two drives (default) makes the floppy driver read the physical cmos for those drives.

floppy=unexpected_interrupts
Print a warning message when an unexpected interrupt is received (default behaviour)

floppy=no_unexpected_interrupts or floppy=L40SX
Don't print a message when an unexpected interrupt is received. This is needed on IBM L40SX laptops in certain video modes. (There seems to be an interaction between video and floppy. The unexpected interrupts only affect performance, and can safely be ignored.)

THE SOUND DRIVER

The sound driver can also accept boot args to override the compiled in values. This is not recommended, as it is rather complex. It is described in the Readme.Linux file, in linux/drivers/sound. It accepts a boot arg of the form:

sound=device1[,device2[,device3...[,device10]]]

where each deviceN value is of the following format 0xTaaaId and the bytes are used as follows:

T - device type: 1=FM, 2=SB, 3=PAS, 4=GUS, 5=MPU401, 6=SB16, 7=SB16-MPU401

aaa - I/O address in hex.

I - interrupt line in hex (i.e 10=a, 11=b, ...)

d - DMA channel.

As you can see it gets pretty messy, and you are better off to compile in your own personal values as recommended. Using a boot arg of `sound=0' will disable the sound driver entirely.

ISDN DRIVERS

The ICN ISDN driver

Syntax:

icn=iobase,membase,icn_id1,icn_id2

where icn_id1,icn_id2 are two strings used to identify the card in kernel messages.

The PCBIT ISDN driver

Syntax:

pcbit=membase1,irq1[,membase2,irq2]

where membaseN is the shared memory base of the N'th card, and irqN is the interrupt setting of the N'th card. The default is IRQ 5 and membase 0xD0000.

The Teles ISDN driver

Syntax:

teles=iobase,irq,membase,protocol,teles_id

where iobase is the i/o port address of the card, membase is the shared memory base address of the card, irq is the interrupt channel the card uses, and teles_id is the unique

ASCII string identifier.

SERIAL PORT DRIVERS

The RISCom/8 Multiport Serial Driver (`riscom8=')

Syntax:

```
riscom=iobase1[,iobase2[,iobase3[,iobase4]]]
```

More details can be found in */usr/src/linux/Documentation/riscom8.txt*.

The DigiBoard Driver (`digi=')

If this option is used, it should have precisely six parameters. Syntax:

```
digi=status,type,altpin,numports,iobase,membase
```

The parameters maybe given as integers, or as strings. If strings are used, then iobase and membase should be given in hexadecimal. The integer arguments (fewer may be given) are in order: status (Enable(1) or Disable(0) this card), type (PC/Xi(0), PC/Xe(1), PC/Xeve(2), PC/Xem(3)), altpin (Enable(1) or Disable(0) alternate pin arrangement), numports (number of ports on this card), iobase (I/O Port where card is configured (in HEX)), membase (base of memory window (in HEX)). Thus, the following two boot prompt arguments are equivalent:

```
digi=E,PC/Xi,D,16,200,D0000
digi=1,0,0,16,0x200,851968
```

More details can be found in */usr/src/linux/Documentation/digiboard.txt*.

The Baycom Serial/Parallel Radio Modem

Syntax:

```
baycom=iobase,irq,modem
```

There are precisely 3 parameters; for several cards, give several `baycom=' commands. The modem parameter is a string that can take one of the values ser12, ser12*, par96, par96*. Here the * denotes that software DCD is to be used, and ser12/par96 chooses between the supported modem types. For more details, see */usr/src/linux/drivers/net/README.baycom*.

Soundcard radio modem driver

Syntax:

```
soundmodem=iobase,irq,dma[,dma2[,serio[,pario]]],0,mode
```

All parameters except the last are integers; the dummy 0 is required because of a bug in the setup code. The mode parameter is a string with syntax hw:modem, where hw is one of sbc, wss, wssfdx and modem is one of afsk1200, fsk9600.

THE LINE PRINTER DRIVER

`lp='

As of kernels newer than 1.3.75, you can tell the printer driver what ports to use and what ports not to use. The latter comes in handy if you don't want the printer driver to claim all available parallel ports, so that other drivers (e.g. PLIP, PPA) can use them instead.

The format of the argument is multiple i/o, IRQ pairs. For example, lp=0x3bc,0,0x378,7 would use the port at 0x3bc in IRQ-less (polling) mode, and use IRQ 7 for the port at 0x378. The port at 0x278 (if any) would not be probed, since autoprobing only takes place in the absence of a `lp=' argument. To disable the printer driver entirely, one can use lp=0.

WDT500/501 driver

Syntax:

```
wdt=io,irq
```

MOUSE DRIVERS

`bmouse=irq'
The busmouse driver only accepts one parameter, that being the hardware IRQ value to be used.

`msmouse=irq'
And precisely the same is true for the msmouse driver.

ATARI mouse setup
atamouse=threshold[,y-threshold]

If only one argument is given, it is used for both x-threshold and y-threshold. Otherwise, the first argument is the x-threshold, and the second the y-threshold. These values must lie between 1 and 20 (inclusive); the default is 2.

VIDEO HARDWARE

`no-scroll'
This option tells the console driver not to use hardware scroll (where a scroll is effected by moving the screen origin in video memory, instead of moving the data). It is required by certain Braille machines.

AUTHORS

Linus Torvalds (and many others)

SEE ALSO

klogd(8), lilo.conf(5), lilo(8), mount(8), rdev(8)

Large parts of this man page have been derived from the Boot Parameter HOWTO (version 1.0.1) written by Paul Gortmaker. Slightly more information may be found in this (or a more recent) HOWTO.

TOC

NAME

glob - Globbing pathnames

DESCRIPTION

Long ago, in Unix V6, there was a program */etc/glob* that would expand wildcard patterns. Soon afterwards this became a shell built-in.

These days there is also a library routine **glob**(3) that will perform this function for a user program.

The rules are as follows (POSIX 1003.2, 3.13).

WILDCARD MATCHING

A string is a wildcard pattern if it contains one of the characters `?', `*' or `['. Globbing is the operation that expands a wildcard pattern into the list of pathnames matching the pattern. Matching is defined by:

A `?' (not between brackets) matches any single character.

A `*' (not between brackets) matches any string, including the empty string.

Character classes

An expression `[...]' where the first character after the leading `[' is not an `!' matches a single character, namely any of the characters enclosed by the brackets. The string enclosed by the brackets cannot be empty; therefore `]' can be allowed between the brackets, provided that it is the

first character. (Thus, `[][!]' matches the three characters `[', `]' and `!'.)

Ranges

There is one special convention: two characters separated by `-' denote a range. (Thus, `[A-Fa-f0-9]' is equivalent to `[ABCDEFabcdef0123456789]'.) One may include `-' in its literal meaning by making it the first or last character between the brackets. (Thus, `[]-]' matches just the two characters `]' and `-', and `[--/]' matches the three characters `-', `.', `/'.)

Complementation

An expression `[!...]' matches a single character, namely any character that is not matched by the expression obtained by removing the first `!' from it. (Thus, `[!]a-]' matches any single character except `]', `a' and `-'.)

One can remove the special meaning of `?', `*' and `[' by preceding them by a backslash, or, in case this is part of a shell command line, enclosing them in quotes. Between brackets these characters stand for themselves. Thus, `[[?*\]' matches the four characters `[', `?', `*' and `\'.

PATHNAMES

Globbing is applied on each of the components of a pathname separately. A `/' in a pathname cannot be matched by a `?' or `*' wildcard, or by a range like `[.-0]'. A range cannot contain an explicit `/' character; this would lead to a syntax error.

If a filename starts with a `.', this character must be matched explicitly. (Thus, `rm *' will not remove .profile, and `tar c *' will not archive all your files; `tar c .' is better.)

EMPTY LISTS

The nice and simple rule given above: `expand a wildcard pattern into the list of matching pathnames' was the original Unix definition. It allowed one to have patterns that expand into an empty list, as in

```
xv -wait 0 *.gif *.jpg
```

where perhaps no *.gif files are present (and this is not an error). However, POSIX requires that a wildcard pattern is left unchanged when it is syntactically incorrect, or the list of matching pathnames is empty. With *bash* on

NAME

hier - Description of the file system hierarchy

DESCRIPTION

A typical Linux system has, among others, the following directories:

/ This is the root directory. This is where the whole tree starts.

/bin This directory contains executable programs which are are needed in single user mode and to bring the system up or repair it.

/boot
Contains static files for the boot loader. This directory only holds the files which are needed during the boot process. The map installer and configuration files should go to */sbin* and */etc*.

/dev Special or device files, which refer to physical devices. See mknod(1).

/dos If both MS-DOS and Linux are run on one computer, this is a typical place to mount a DOS file system.

/etc Contains configuration files which are local to the machine. Some larger software packages, like X11, can have their own subdirectories below */etc*. Site-wide configuration files may be placed here or in */usr/etc*. Nevertheless, programs should always look for these files in */etc* and you may have links for these files to */usr/etc*.

/etc/skel

When a new user account is created, files from this directory are usually copied into the user's home directory.

/etc/X11
Configuration files for the X11 window system.

/home
On machines with home directories for users, these are usually beneath this directory, directly or not. The structure of this directory depends on local adminins-tration decisions.

/lib This directory should hold those shared libraries that are necessary to boot the system and to run the com-mands in the root filesystem.

/mnt is a mount point for temporarily mounted filesystems

/proc
This is a mount point for the *proc* filesystem, which provides information about running processes and the kernel. This pseudo-file system is described in more detail in *proc(5)*.

/sbin
Like */bin*, this directory holds commands needed to boot the system, but which are usually not executed by nor-mal users.

/tmp This directory contains temporary files which may be deleted with no notice, such as by a regular job or at system boot up.

/usr This directory is usually mounted from a seperate par-tition. It should hold only sharable, read-only data, so that it can be mounted by various machines running Linux.

/usr/X11R6
The X-Window system, version 11 release 6.

/usr/X11R6/bin
Binaries which belong to the X-Windows system; often, there is a symbolic link from the more traditional */usr/bin/X11* to here.

/usr/X11R6/lib
Data files associated with the X-Windows system.

/usr/X11R6/lib/X11
These contain miscellaneous files needed to run X; Often, there is a symbolic link from */usr/lib/X11* to this directory.

/usr/X11R6/include/X11
Contains include files needed for compiling programs using the X11 window system. Often, there is a symbolic link from */usr/inlcude/X11* to this directory.

/usr/bin
This is the primary directory for executable programs. Most programs executed by normal users which are not needed for booting or for repairing the system and which are not installed locally should be placed in this directory.

/usr/bin/X11
is the traditional place to look for X11 executables; on Linux, it usually is a symbolic link to */usr/X11R6/bin*.

/usr/dict
This directory holds files containing word lists for spell checkers.

/usr/etc
Site-wide configuration files to be shared between several machines may be stored in this directory. However, commands should always reference those files using the */etc* directory. Links from files in */etc* should point to the appropriate files in */usr/etc*.

/usr/include
Include files for the C compiler.

/usr/include/X11
Include files for the C compiler and the X-Windows system. This is usually a symbolic link to */usr/X11R6/include/X11*.

/usr/include/asm

Include files which declare some assembler functions. This used to be a symbolic link to */usr/src/linux/include/asm*, but this isn't the case in Debian or libc6 based systems.

/usr/include/linux
This contains information which may change from system release to system release and used to be a symbolic link to */usr/src/linux/include/linux* to get at operating system specific information. Debian systems don't do this and use headers from a known good kernel version, provided in the libc*-dev package.

*/usr/include/***g**++
Include files to use with the GNU C++ compiler.

/usr/lib
Object libraries, including dynamic libraries, plus some executables which usually are not invoked directly. More complicated programs may have whole subdirectories there.

/usr/lib/X11
The usual place for data files associated with X programs, and configuration files for the X system itself. On Linux, it usually is a symbolic link to */usr/X11R6/lib/X11*.

/usr/lib/gcc-lib
contains executables and include files for the GNU C compiler, **gcc**(1).

/usr/lib/groff
Files for the GNU groff document formatting system.

/usr/lib/uucp
Files for **uucp**(1).

/usr/lib/zoneinfo
Files for timezone information.

/usr/local
This is where programs which are local to the site typically go in.

/usr/local/bin

Binaries for programs local to the site go there.

/usr/local/doc
Local documnetation

/usr/local/etc
Configuration files associated with locally installed programs go there.

/usr/local/lib
Files associated with locally installed programs go there.

/usr/local/info
Info pages associated with locally installed programs go there.

/usr/local/man
Manpages associated with locally installed programs go there.

/usr/local/sbin
Locally installed programs for system admininstration.

/usr/local/src
Source code for locally installed software.

/usr/man
Manpages go in there, into their subdirectories.

/usr/man/<locale>/man[**1**-**9**]
These directories contain manual pages which are in source code form. Systems which use a unique language and code set for all manual pages may omit the *<locale>* substring.

/usr/sbin
This directories contains program binaries for system admininstration which are not essentail for the boot process, for mounting */usr*, or for system repair.

/usr/src
Source files for different parts of the system.

/usr/src/linux
This contains the sources for the kernel of the operat-

ing system itself.

/usr/tmp
An alternative place to store temporary files; This should be a link to */var/tmp*. This link is present only for compatibility reasons and shouldn't be used.

/var This directory contains files which may change in size, such as spool and log files.

/var/adm
This directory is superseded by */var/log* and should be a symbolic link to */var/log*.

/var/backups
This directory is used to save backup copies of impor-tant system files.

/var/catman/cat[**1**-**9**]
These directories contain preformatted manual pages according to their manpage section.

/var/lock
Lock files are plaed in this directory. The naming convention for device lock files is *LCK..<device>* where *<device>* is the device's name in the filesystem. The format used is that of HDU UUCP lock files, i.e. lock files contain a PID as a 10-byte ASCII decimal number, followed by a newline character.

/var/log
Miscelanous log files.

/var/preserve
This is where **vi**(1) saves edit sessions so they can be restored later.

/var/run
Run-time varaible files, like files holding process identifiers (PIDs) and logged user information (*utmp*). Files in this directory are usually cleared when the system boots.

/var/spool
Spooled (or queued) files for various programs.

/var/spool/at
Spooled jobs for **at**(1).

/var/spool/cron
Spooled jobs for **cron**(1).

/var/spool/lpd
Spooled files for printing.

/var/spool/mail
User's mailboxes.

/var/spool/smail
Spooled files for the **smail**(1) mail delivery program.

/var/spool/news
Spool directory for the news subsystem.

/var/spool/uucp
Spooled files for **uucp**(1).

/var/tmp
Like */tmp*, this directory holds temporary files stored for an unspecified duration.

CONFORMS TO

The Linux filesystem standard, Release 1.2

BUGS

This list is not exhaustive; different systems may be configured differently.

SEE ALSO

find(1), **ln**(1), **mount**(1), **proc**(5), The Linux Filesystem Standard

NAME

intro - Introduction to miscellany section

DESCRIPTION

This chapter describes miscellaneous things such as nroff macro packages, tables, C header files, the file hierarchy, general concepts, and other things which don't fit anywhere else.

AUTHORS

Look at the header of the manual page for the author(s) and copyright conditions. Note that these can be different from page to page!

TOC

NAME

locale - Description of multi-language support

SYNOPSIS

#include <locale.h>

DESCRIPTION

A locale is a set of language and cultural rules. These cover aspects such as language for messages, different character sets, lexigraphic conventions, etc. A program needs to be able to determine its locale and act accordingly to be portable to different cultures.

The header **<locale.h>** declares data types, functions and macros which are useful in this task.

The functions it declares are **setlocale()** to set the current locale, and **localeconv()** to get information about number formatting.

There are different categories for local information a program might need; they are declared as macros. Using them as the first argument to the **setlocale()** function, it is possible to set one of these to the desired locale:

LC_COLLATE
This is used to change the behaviour of the functions **strcoll()** and **strxfrm()**, which are used to compare strings in the local alphabet. For example, the German sharp s is sorted as "ss".

LC_CTYPE
This changes the behaviour of the character handling and classification functions, such as **isupper()** and **toupper()**, and the multi-byte character functions such as **mblen()** or **wctomb()**.


TOC

LC_MONETARY
changes the information returned by **localeconv()** which describes the way numbers are usually printed, with details such as decimal point versus decimal comma. This information is internally used by the functions **strfmon() .**

LC_MESSAGES
changes the language messages are displayed in and how an affirmative or negative answer looks like. The GNU C-library contains the **rpmatch()** function to ease the use of these information.

LC_NUMERIC
changes the informations used by the **printf()** and **scanf()** family of functions, when they are advised to use the locale-settings. This information can also be read with the **localeconv()** function.

LC_TIME
changes the behaviour of the **strftime()** function to display the current time in a locally acceptable form; for example, most of Europe uses a 24-hour clock vs. the US' 12-hour clock.

LC_ALL
All of the above.

If the second argument to **setlocale()** is empty string, for the default locale, it is determined using the following steps:

1. If there is a non-null environment variable **LC_ALL**, the value of **LC_ALL** is used.

2. If an environment variable with the same name as one of the categories above exists and is non-null, its value is used for that category.

3. If there is a non-null environment variable **LANG**, the value of **LANG** is used.

Values about local numeric formatting is made available in a **struct lconv** returned by the **localeconv()** function, which has the following declaration:

```
struct lconv
{
  /* Numeric (non-monetary) information.  */

  char *decimal_point;        /* Decimal point character.  */
  char *thousands_sep;        /* Thousands separator.  */
  /* Each element is the number of digits in each group;
```


```
      elements with higher indices are farther left.
      An element with value CHAR_MAX means that no further grouping is d
      An element with value 0 means that the previous element is used
      for all groups farther left.  */
   char *grouping;

   /* Monetary information.  */

   /* First three chars are a currency symbol from ISO 4217.
      Fourth char is the separator.  Fifth char is ' '.  */
   char *int_curr_symbol;
   char *currency_symbol; /* Local currency symbol.  */
   char *mon_decimal_point;    /* Decimal point character.  */
   char *mon_thousands_sep;    /* Thousands separator.  */
   char *mon_grouping;         /* Like `grouping' element (above).  */
   char *positive_sign;        /* Sign for positive values.  */
   char *negative_sign;        /* Sign for negative values.  */
   char int_frac_digits;       /* Int'l fractional digits.  */
   char frac_digits;      /* Local fractional digits.  */
   /* 1 if currency_symbol precedes a positive value, 0 if succeeds.  */
   char p_cs_precedes;
   /* 1 if a space separates currency_symbol from a positive value.  */
   char p_sep_by_space;
   /* 1 if currency_symbol precedes a negative value, 0 if succeeds.  */
   char n_cs_precedes;
   /* 1 if a space separates currency_symbol from a negative value.  */
   char n_sep_by_space;
   /* Positive and negative sign positions:
      0 Parentheses surround the quantity and currency_symbol.
      1 The sign string precedes the quantity and currency_symbol.
      2 The sign string succeeds the quantity and currency_symbol.
      3 The sign string immediately precedes the currency_symbol.
      4 The sign string immediately succeeds the currency_symbol.  */
   char p_sign_posn;
   char n_sign_posn;
};
```

CONFORMS TO

POSIX.1

SEE ALSO

setlocale(3), **localeconv**(3), **locale**(1), **rpmatch**(3),

strfmon(3), **strcoll**(3), **strftime**(3)

NAME

mailaddr - mail addressing description

DESCRIPTION

This manual page gives a brief introduction to SMTP mail addresses, as used on the Internet. These addresses are in the general format

```
user@domain
```

where a domain is a hierarchical dot separated list of sub-domains. For example, the addresses

```
eric@monet.berkeley.edu
Eric Allman <eric@monet.berkeley.edu>
eric@monet.berkeley.edu (Eric Allman)
```

are valid forms of the same address.

The domain part (``monet.berkeley.edu'') may be the name of an internet host, or it may be a logical mail address. The domain part is not case sensitive.

The local part (``eric'') is often a user name, but its meaning is defined by the local software. It can be case sensitive, but usually isn't. If you see a local-part that looks like garbage, it is usually because of a gateway between an internal e-mail system and the net, here are some examples:

```
"surname/admd=telemail/c=us/o=hp/prmd=hp"@some.where
USER%SOMETHING@some.where
machine!machine!name@some.where
I2461572@some.where
```

(These are, respectively, an X.400 gateway, a gateway to an arbitrary inernal mail system that lacks proper internet support, an UUCP gateway, and the last one is just boring username policy.)

The real-name part (``Eric Allman'') can either be placed first, outside <>, or last, inside (). (Strictly speaking the two aren't the same, but the difference is outside the scope of this page.) The name may have to be quoted using "" if it contains certain characters, most commonly ``.'':

"Eric P. Allman" <eric@monet.berkeley.edu>

Abbreviation.

Many mail systems let users abbreviate the domain name. For instance, users at berkeley.edu may get away with ``eric@monet'' to send mail to Eric Allman. *This behavior is deprecated*.

Route-addrs.

Under some circumstances it may be necessary to route a message through several hosts to get it to the final destination. Normally this happens automatically and invisibly, but sometimes not, particularly with old and broken software. Addresses which show these relays are termed ``route-addrs.'' These use the syntax:

<@hosta,@hostb:user@hostc>

This specifies that the message should be sent to hosta, from there to hostb, and finally to hostc. Some hosts disregard route-addrs and send directly to hostc.

Route-addrs occur frequently on return addresses, since these are generally augmented by the software at each host. It is generally possible to ignore all but the ``user@hostc'' part of the address to determine the actual sender.

Postmaster.

Every site is required to have a user or user alias designated ``postmaster'' to which problems with the mail system may be addressed. The ``postmaster'' address is not case sensitive.

FREQUENTLY ASKED QUESTIONS

rtfm.mit.edu and many mirrors store a collection of FAQs. Please find and use a nearby FAQ archive; there are dozens or hundreds around the world. *mail/inter-network-guide* explains how to send mail between many different networks. *mail/country-codes* lists the top level domains (e.g. ``no'' is Norway and ``ea'' is Eritrea). *mail/college-email/part** gives some useful tips on how to locate e-mail addresses.

FILES

/etc/aliases
~/.forward

SEE ALSO

binmail(1), **mail**(1), **mconnect**(1), **forward**(5), **aliases**(5), **sendmail**(8), **vrfy**(8), RFC822 (Standard for the Format of Arpa Internet Text Messages).

TOC

NAME

man - macros to format man pages

SYNOPSIS

groff **-Tascii** **-man** *file* ...

groff **-Tps** **-man** *file* ...

man [*section*] *title*

DESCRIPTION

This manual page explains the **groff tmac.an** macro package. This macro package should be used by developers when writing or porting man pages for Linux. It is fairly compatible with other versions of this macro package, so porting man pages should not be a major problem (exceptions include the NET-2 BSD release, which uses a totally different macro package).

Note that NET-2 BSD man pages can be used with **groff** simply by specifying the **-mdoc** option instead of the **-man** option. Using the **-mandoc** option is, however, recommended, since this will automatically detect which macro package is in use.

PREAMBLE

The first command in a man page should be

.TH *title section date source manual*,

where:

title The title of the man page (e.g., *MAN*).

section The section number the man page should be placed in (e.g., **7**).

date The date of the last revision-remember to change this every time a change is made to the man page, since this is the most general way of doing version control.

source The source of the command.

For binaries, use something like: *GNU*, *NET-***2**, *SLS Distribution*,

For system calls, use the version of the kernel that you are currently looking at: *Linux 0.99.11*.

For library calls, use the source of the function: *GNU*, *BSD 4.3*, *Linux DLL 4.4*.**1**.

manual The title of the manual (e.g., *Linux Programmer*'**s** *Manual*).

The manual sections are traditionally defined as follows:

1 Commands
Those commands that can be executed by the user from within a shell.

2 System calls
Those functions which must be performed by the kernel.

3 Library calls

Most of the *libc* functions, such as **sort**(3))

4 Special files
Files found in */dev)*

5 File formats and conventions
The format for */etc/passwd* and other human-readable files.

6 Games

7 Macro packages and conventions
A description of the standard file system lay-out, this man page, and other things.

8 System management commands
Commands like **mount**(8), which only *root* can execute.

9 Kernel routines
This is a non-standard manual section and is included because the source code to the Linux kernel is freely available under the GNU Public License and many people are working on changes to the kernel)

FONTS

Although there are many arbitrary conventions for man pages in the UNIX world, the existence of several hundred Linux-specific man pages defines our standards:

For functions, the arguments are always specified using italics, *even in the SYNOPSIS section*, where the rest of the function is specified in bold:
int myfunction(int *argc*, **char *****argv)*);

Filenames are always in italics (e.g., */usr/include/stdio.h)*, except in the SYNOPSIS section, where included files are in bold (e.g., **#include <stdio.h>**).

Special macros, which are usually in upper case, are in bold (e.g., **MAXINT**).

When enumerating a list of error codes, the codes are in bold (this list usually uses the **.TP** macro).

Any reference to another man page (or to the subject of the current man page) is in bold. If the manual section number is given, it is given in roman, without any spaces (e.g., **man**(7)).

The commands to select the type face are given below:

.B Bold

.BI Bold alternating with italics

.BR Bold alternating with Roman

.I Italics

.IB Italics alternating with bold

.IR Italics alternating with Roman

.RB Roman alternating with bold

.RI Roman alternating with italics

.SB Small alternating with bold

.SM Small

Traditionally, each command can have up to six arguments, but the GNU version seems to remove this limitation. Arguments are delimited by spaces. Double quotes can be used to specify an argument which contains spaces. All of the arguments will be printed next to each other without intervening spaces, so that the **.BR** command can be used to specify a word in bold followed by a mark of punctuation in Roman.

SECTIONS

Sections are started with **.SH** followed by the heading name. If the name contains spaces and appears on the same line as **.SH**, then place the heading in double quotes. Traditional headings include: NAME, SYNOPSIS, DESCRIPTION, OPTIONS, FILES, SEE ALSO, DIAGNOSTICS, BUGS, and AUTHOR. The only required heading is *NAME*, which should be followed on the next line by a one line description of the program:

```
.SH NAME
chess \- the game of chess
```

It is extremely important that this format is followed, and that there is a backslash before the single dash which follows the command name. This syntax is used by the **makewhatis**(8) program to create a database of short command descriptions for the **whatis**(1) and **apropos**(1) commands.

OTHER MACROS

Other macros include the following:

.DT Default tabs

.HP Begin hanging indent

.IP Begin paragraph with hanging tag. This is the same as **.TP**, except the tag is given on the same line, not on the following line.

.LP Same as **.PP**

.PD Set interparagraph distance to argument

.PP Begin a new paragraph

.RE End relative indent (indented paragraph)

.RS Start relative indent (indented paragraph)

.SS Subheading (like **.SH**, but used for a subsection)

.TP Begin paragraph with hanging tag. The tag is given on the next line. This command is similar to **.IP**

FILES

/usr/local/lib/groff/tmac/tmac.an
/usr/man/whatis

SEE ALSO

groff(1), **man**(1), **whatis**(1), **apropos**(1), **makewhatis**(8)

NAME

regex - POSIX 1003.2 regular expressions

DESCRIPTION

Regular expressions (``RE''s), as defined in POSIX 1003.2, come in two forms: modern REs (roughly those of *egrep*; 1003.2 calls these ``extended'' REs) and obsolete REs (roughly those of *ed*; 1003.2 ``basic'' REs). Obsolete REs mostly exist for backward compatibility in some old programs; they will be discussed at the end. 1003.2 leaves some aspects of RE syntax and semantics open; `-' marks decisions on these aspects that may not be fully portable to other 1003.2 implementations.

A (modern) RE is one- or more non-empty- *branches*, separated by `|'. It matches anything that matches one of the branches.

A branch is one- or more *pieces*, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an *atom* possibly followed by a single- `*', `+', `?', or *bound*. An atom followed by `*' matches a sequence of 0 or more matches of the atom. An atom followed by `+' matches a sequence of 1 or more matches of the atom. An atom followed by `?' matches a sequence of 0 or 1 matches of the atom.

A *bound* is `{' followed by an unsigned decimal integer, possibly followed by `,' possibly followed by another unsigned decimal integer, always followed by `}'. The integers must lie between 0 and RE_DUP_MAX (255-) inclusive, and if there are two of them, the first may not exceed the second. An atom followed by a bound containing one integer **i** and no

comma matches a sequence of exactly **i** matches of the atom. An atom followed by a bound containing one integer **i** and a comma matches a sequence of **i** or more matches of the atom. An atom followed by a bound containing two integers **i** and **j** matches a sequence of **i** through **j** (inclusive) matches of the atom.

An atom is a regular expression enclosed in `()' (matching a match for the regular expression), an empty set of `()' (matching the null string)**-**, a *bracket expression* (see below), `.' (matching any single character), `^' (matching the null string at the beginning of a line), `$' (matching the null string at the end of a line), a `\' followed by one of the characters `^.[$()|*+?{\' (matching that character taken as an ordinary character), a `\' followed by any other character**-** (matching that character taken as an ordinary character, as if the `\' had not been present**-**), or a single character with no other significance (matching that character). A `{' followed by a character other than a digit is an ordinary character, not the beginning of a bound**-**. It is illegal to end an RE with `\'.

A *bracket expression* is a list of characters enclosed in `[]'. It normally matches any single character from the list (but see below). If the list begins with `^', it matches any single character (but see below) *not* from the rest of the list. If two characters in the list are separated by `-', this is shorthand for the full *range* of characters between those two (inclusive) in the collating sequence, e.g. `[0-9]' in ASCII matches any decimal digit. It is illegal**-** for two ranges to share an endpoint, e.g. `a-c-e'. Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.

To include a literal `]' in the list, make it the first character (following a possible `^'). To include a literal `-', make it the first or last character, or the second endpoint of a range. To use a literal `-' as the first endpoint of a range, enclose it in `[.' and `.]' to make it a collating element (see below). With the exception of these and some combinations using `[' (see next paragraphs), all other special characters, including `\', lose their special significance within a bracket expression.

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were

a single character, or a collating-sequence name for either) enclosed in `[.' and `.]' stands for the sequence of characters of that collating element. The sequence is a single element of the bracket expression's list. A bracket expression containing a multi-character collating element can thus match more than one character, e.g. if the collating sequence includes a `ch' collating element, then the RE `[[.ch.]]*c' matches the first five characters of `chchcc'.

Within a bracket expression, a collating element enclosed in `[=' and `=]' is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. (If there are no other equivalent collating elements, the treatment is as if the enclosing delimiters were `[.' and `.]'.) For example, if o and ^ are the members of an equivalence class, then `[[=o=]]', `[[=^=]]', and `[o^]' are all synonymous. An equivalence class may not- be an endpoint of a range.

Within a bracket expression, the name of a *character class* enclosed in `[:' and `:]' stands for the list of all characters belonging to that class. Standard character class names are:

alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

These stand for the character classes defined in *ctype*(3). A locale may provide others. A character class may not be used as an endpoint of a range.

There are two special cases- of bracket expressions: the bracket expressions `[[:<:]]' and `[[:>:]]' match the null string at the beginning and end of a word respectively. A word is defined as a sequence of word characters which is neither preceded nor followed by word characters. A word character is an *alnum* character (as defined by *ctype*(3)) or an underscore. This is an extension, compatible with but not specified by POSIX 1003.2, and should be used with caution in software intended to be portable to other systems.

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, it matches the longest.

Subexpressions also match the longest possible substrings, subject to the constraint that the whole match be as long as possible, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that higher-level subexpressions thus take priority over their lower-level component subexpressions.

Match lengths are measured in characters, not collating elements. A null string is considered longer than no match at all. For example, `bb*' matches the three middle characters of `abbbc', `(wee|week)(knights|nights)' matches all ten characters of `weeknights', when `(.*).*' is matched against `abc' the parenthesized subexpression matches all three characters, and when `(a*)*' is matched against `bc' both the whole RE and the parenthesized subexpression match the null string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, e.g. `x' becomes `[xX]'. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that (e.g.) `[x]' becomes `[xX]' and `[^x]' becomes `[^xX]'.

No particular limit is imposed on the length of REs-. Programs intended to be portable should not employ REs longer than 256 bytes, as an implementation can refuse to accept such REs and remain POSIX-compliant.

Obsolete (``basic'') regular expressions differ in several respects. `|', `+', and `?' are ordinary characters and there is no equivalent for their functionality. The delimiters for bounds are `\{' and `\}', with `{' and `}' by themselves ordinary characters. The parentheses for nested subexpressions are `\(' and `\)', with `(' and `)' by themselves ordinary characters. `^' is an ordinary character except at the beginning of the RE or- the beginning of a parenthesized subexpression, `$' is an ordinary character except at the end of the RE or- the end of a parenthesized subexpression, and `*' is an ordinary character if it appears at the beginning of the RE or the beginning of a parenthesized subexpression (after a possible leading `^'). Finally, there is one new type of atom, a *back reference*:

`\' followed by a non-zero decimal digit **d** matches the same sequence of characters matched by the **d**th parenthesized subexpression (numbering subexpressions by the positions of their opening parentheses, left to right), so that (e.g.) `\([bc]\)\1' matches `bb' or `cc' but not `bc'.

SEE ALSO

regex(3)

POSIX 1003.2, section 2.8 (Regular Expression Notation).

BUGS

Having two kinds of REs is a botch.

The current 1003.2 spec says that `)' is an ordinary character in the absence of an unmatched `('; this was an unintentional result of a wording error, and change is likely. Avoid relying on it.

Back references are a dreadful botch, posing major problems for efficient implementations. They are also somewhat vaguely defined (does `a\(\(b\)*\2\)*d' match `abbbd'?). Avoid using them.

1003.2's specification of case-independent matching is vague. The ``one case implies all cases'' definition given above is current consensus among implementors as to the right interpretation.

The syntax for word boundaries is incredibly ugly.

AUTHOR

TOC

This page was taken from Henry Spencer's regex package.

TOC

NAME

signal - list of available signals

DESCRIPTION

Linux supports the signals listed below. Several signal numbers are architecture dependent. First the signals described in POSIX.1.

Signal	Value	Action	Comment
SIGHUP	1	A	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	A	Interrupt from keyboard
SIGQUIT	3	A	Quit from keyboard
SIGILL	4	A	Illegal Instruction
SIGABRT	6	C	Abort signal from *abort*(3)
SIGFPE	8	C	Floating point exception
SIGKILL	9	AEF	Kill signal
SIGSEGV	11	C	Invalid memory reference
SIGPIPE	13	A	Broken pipe: write to pipe with no readers
SIGALRM	14	A	Timer signal from *alarm*(2)
SIGTERM	15	A	Termination signal
SIGUSR1	30,10,16	A	User-defined signal 1
SIGUSR2	31,12,17	A	User-defined signal 2
SIGCHLD	20,17,18	B	Child stopped or terminated
SIGCONT	19,18,25		Continue if stopped
SIGSTOP	17,19,23	DEF	Stop process
SIGTSTP	18,20,24	D	Stop typed at tty
SIGTTIN	21,21,26	D	tty input for background process
SIGTTOU	22,22,27	D	tty output for background process

Next various other signals.

Signal	Value	Action	Comment

Signal	Value	Action	Comment
SIGTRAP	5	CG	Trace/breakpoint trap
SIGIOT	6	CG	IOT trap. A synonym for SIGABRT
SIGEMT	7,-,7	G	
SIGBUS	10,7,10	AG	Bus error
SIGSYS	12,-,12	G	Bad argument to routine (SVID)
SIGSTKFLT	-,16,-	AG	Stack fault on coprocessor
SIGURG	16,23,21	BG	Urgent condition on socket (4.2 BSD)
SIGIO	23,29,22	AG	I/O now possible (4.2 BSD)
SIGPOLL		AG	A synonym for SIGIO (System V)
SIGCLD	-,-,18	G	A synonym for SIGCHLD
SIGXCPU	24,24,30	AG	CPU time limit exceeded (4.2 BSD)
SIGXFSZ	25,25,31	AG	File size limit exceeded (4.2 BSD)
SIGVTALRM	26,26,28	AG	Virtual alarm clock (4.2 BSD)
SIGPROF	27,27,29	AG	Profile alarm clock
SIGPWR	29,30,19	AG	Power failure (System V)
SIGINFO	29,-,-	G	A synonym for SIGPWR
SIGLOST	-,-,-	AG	File lock lost
SIGWINCH	28,28,20	BG	Window resize signal (4.3 BSD, Sun)
SIGUNUSED	-,31,-	AG	Unused signal (Here - denotes that a signal is absent; there where three values are given, the first one is usually valid for alpha and sparc, the middle one for i386 and ppc, the last one for mips. Signal 29 is SIGINFO / SIGPWR on an alpha but SIGLOST on a sparc.)

The letters in the "Action" column have the following meanings:

A Default action is to terminate the process.

B Default action is to ignore the signal.

C Default action is to dump core.

D Default action is to stop the process.

E Signal cannot be caught.

F Signal cannot be ignored.

G Not a POSIX.1 conformant signal.

CONFORMING TO

POSIX.1

BUGS

SIGIO and SIGLOST have the same value. The latter is commented out in the kernel source, but the build process of some software still thinks that signal 29 is SIGLOST.

SEE ALSO

kill(1), **kill**(2), **setitimer**(2)

TOC

NAME

suffixes - list of file suffixes

DESCRIPTION

It is customary to indicate the contents of a file with the file suffix, which consists of a period, followed by one or more letters. Many standard utilities, such as compilers, use this to recognize the type of file they are dealing with. The **make**(1) utility is driven by rules based on file suffix.

Following is a list of suffixes which are likely to be found on a Linux system.

l | l _ | _ lI | l . Suffix File type
,v files for RCS (Revision Control System)
- backup file
.C C++ source code
.F Fortran source with **cpp**(1) directives or file compressed using freeze
.S assembler source with **cpp**(1) directives
.Y file compressed using yabba
.Z file compressed using **compress**(1)
.[0-9]+gf TeX generic font files
.[0-9]+pk TeX packed font files
.[1-9] manual page for the corresponding section
.[1-9][a-z] manual page for section plus subsection
.a static object code library
.ad X application default resource file
.afm PostScript font metrics
.al Perl autoload file
.am **automake**(1) input file
.arc **arc**(1) archive
.arj **arj**(1) archive
.asc PGP ASCII-armoured data

.asm (GNU) assember source file
.au Audio sound file
.aux LaTeX auxiliary file
.avi (msvideo) movie
.awk AWK language program
.b LILO boot loader image
.bak backup file
.bash **bash**(1) shell script
.bb basic block list data produced by gcc -ftest-coverage
.bbg basic block graph data produced by gcc -ftest-coverage
.bbl BibTeX output
.bdf X font file
.bib TeX bibliographic database, BibTeX input
.bm bitmap source
.bmp bitmap
.c C source
.cat message catalog files
.cc C++ source
.cf configuration file
.cfg configuration file
.cgi WWW content generating script or program
.class Java compiled byte-code
.conf configuration file
.config configuration file
.cpp equivalent to *.cc*
.csh **csh**(1) shell script
.cxx equivalent to *.cc*
.dat data file
.deb Debian software package
.def Modula-2 source for definition modules
.def other definition files
.desc initial part of mail message unpacked with munpack
.diff file differences (diff(1) command output)
.dir dbm data base directory file
.doc documentation file
.dtx LaTeX package source file
.dvi TeX's device independent output
.el Emacs-Lisp source
.elc compiled Emacs-Lisp source
.eps encapsulated PostScript
.f Fortran source
.fas pre-compiled Common-Lisp
.fi Fortran include files
.fig FIG image file (used by **xfig**(1))
.fmt TeX format file

.gif Compuserve Graphics Image File format
.gmo GNU format message catalog
.gsf Ghostscript fonts
.gz file compressed using **gzip**(1)
.h C or C++ header files
.help help file
.hf equivalent to *.help*
.hlp equivalent to *.help*
.htm poor man's *.html*
.html HTML document used with the World Wide Web
.hqx 7-bit encoded Macintosh file
.i C source after preprocessing
.icon bitmap source
.idx reference or datum-index file for hypertext or database system
.image bitmap source
.in configuration template, especially for GNU Autoconf
.info files for the Emacs info browser
.info-[0-9]+ splitted info files
.ins LaTeX package install file for docstrip
.java a Java source file
.jpeg Joint Photographic Experts Group format
.jpg poor man's *.jpeg*
.l equivalent to *.lex* or *.lisp*
.lex **lex**(1) or **flex**(1) files
.lha lharc archive
.lib Common-Lisp library
.lisp Lisp source
.ln files for use with **lint**(1)
.log log file, in particular produced by TeX
.lsm Linux Software Map entry
.lsp Common-Lisp source
.lzh lharc archive
.m4 **m4**(1) source
.mac macro files for various programs
.man manual page (usually source rather than formatted)
.map map files for various programs
.me Nroff source using the me macro package
.mf Metafont (font generator for TeX) source
.mm sources for **groff**(1) in mm - format
.mo Message catalog binary file
.mod Modula-2 source for implementation modules
.mov (quicktime) movie
.mp Metapost source
.mpeg movie file
.o object file

.old old or backup file
.orig backup (original) version of a file, from **patch**(1)
.out output file, often executable program (a.out)
.p Pascal source
.pag dbm data base data file
.patch file differences for **patch**(1)
.pbm portable bitmap format
.pcf X11 font files
.pdf Adobe Portable Data Format (use Acrobat/**acroread** or **xpdf**)
.perl Perl source
.pfa PostScript font definition files, ASCII format
.pfb PostScript font definition files, binary format
.pgm portable greymap format
.pgp PGP binary data
.ph Perl header file
.pid File to store daemon pid (e.g. crond.pid)
.pl TeX property list file or Perl library file
.pm Perl module
.png Portable Network Graphics file
.po Message catalog source
.pod **perldoc**(1) file
.ppm portable pixmap format
.pr bitmap source
.ps PostScript file
.py Python source
.pyc compiled python
.qt quicktime movie
.r RATFOR source (obsolete)
.rej patches that **patch**(1) couldn't apply
.rpm RedHat software package
.rtf Rich Text Format file
.rules rules for something
.s assembler source
.sa stub libraries for a.out shared libraries
.sc **sc**(1) spreadsheed commands
.sgml SGML source file
.sh **sh**(1) scripts
.shar archive created by the **shar**(1) utility
.so Shared library or dynamically loadable object
.sql SQL source
.sqml SQML schema or query program
.sty LaTeX style files
.sym Modula-2 compiled definition modules
.tar archive created by the **tar**(1) utility
.tar.Z tar(1) archive compressed with **compress**(1)

.tar.gz tar(1) archive compressed with **gzip**(1)
.taz tar(1) archive compressed with **compress**(1)
.tex TeX or LaTeX source
.texi equivalent to .*texinfo*
.texinfo Texinfo documentation source
.text text file
.tfm TeX font metric file
.tgz tar archive compressed with **gzip**(1)
.tif poor man's .*tiff*
.tiff Tagged Image File Format
.tk tcl/tk script
.tmp temporary file
.tmpl template files
.txt equivalent to .*text*
.uu equivalent to .*uue*
.uue binary file encoded with **uuencode**(1)
.vf TeX virtual font file
.vpl TeX virtual property list file
.w Silvio Levi's CWEB
.wav wave sound file
.web Donald Knuth's WEB
.xbm X11 bitmap source
.xpm X11 pixmap source
.xs Perl xsub file produced by h2xs
.y **yacc**(1) or **bison**(1) (parser generator) files
.z File compressed using **pack**(1) (or an old **gzip**(1))
.zip **zip**(1) archive
.zoo **zoo**(1) archive
~ Emacs or **patch**(1) backup file
rc startup (`run control') file, e.g. .*newsrc*

CONFORMS TO

General UNIX conventions.

BUGS

This list is not exhaustive.

SEE ALSO

file(1), **make**(1)

TOC

TOC

NAME

Unicode - the unified 16-bit super character set

DESCRIPTION

The international standard **ISO 10646** defines the **Universal Character Set (UCS)**. **UCS** contains all characters of all other character set standards. It also guarantees **round-trip compatibility**, i.e., conversion tables can be built such that no information is lost when a string is converted from any other encoding to **UCS** and back.

UCS contains the characters required to represent almost all known languages. This includes apart from the many languages which use extensions of the Latin script also the following scripts and languages: Greek, Cyrillic, Hebrew, Arabic, Armenian, Gregorian, Japanese, Chinese, Hiragana, Katakana, Korean, Hangul, Devangari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayam, Thai, Lao, Bopomofo, and a number of others. Work is going on to include further scripts like Tibetian, Khmer, Runic, Ethiopian, Hieroglyphics, various Indo-European languages, and many others. For most of these latter scripts, it was not yet clear how they can be encoded best when the standard was published in 1993. In addition to the characters required by these scripts, also a large number of graphical, typographical, mathematical and scientific symbols like those provided by TeX, PostScript, MS-DOS, Macintosh, Videotext, OCR, and many word processing systems have been included, as well as special codes that guarantee round-trip compatibility to all other existing character set standards.

The **UCS** standard (ISO 10646) describes a 31-bit character set architecture, however, today only the first 65534 code positions (0x0000 to 0xfffd), which are called the **Basic Multilingual Plane (BMP)**, have been assigned characters, and

it is expected that only very exotic characters (e.g. Hieroglyphics) for special scientific purposes will ever get a place outside this 16-bit BMP.

The **UCS** characters 0x0000 to 0x007f are identical to those of the classic **US-ASCII** character set and the characters in the range 0x0000 to 0x00ff are identical to those in the **ISO 8859-1 Latin-1** character set.

COMBINING CHARACTERS

Some code points in **UCS** have been assigned to **combining characters**. These are similar to the non-spacing accent keys on a typewriter. A combining character just adds an accent to the previous character. The most important accented characters have codes of their own in **UCS**, however, the combining character mechanism allows to add accents and other diacritical marks to any character. The combining characters always follow the character which they modify. For example, the German character Umlaut-A ("Latin capital letter A with diaeresis") can either be represented by the precomposed **UCS** code 0x00c4, or alternatively as the combination of a normal "Latin capital letter A" followed by a "combining diaeresis": 0x0041 0x0308.

IMPLEMENTATION LEVELS

As not all systems are expected to support advanced mechanisms like combining characters, ISO 10646 specifies the following three implementation levels of **UCS:**

Level 1 Combining characters and Hangul Jamo characters (a special, more complicated encoding of the Korean script, where Hangul syllables are coded as two or three subcharacters) are not supported.

Level 2 Like level 1, however in some scripts, some combining characters are now allowed (e.g. for Hebrew,

Arabic, Devangari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugo, Kannada, Malayalam, Thai and Lao).

Level 3 All **UCS** characters are supported.

The Unicode 1.1 standard published by the Unicode Consortium contains exactly the **UCS Basic Multilingual Plane** at implementation level 3, as described in ISO 10646. Unicode 1.1 also adds some semantical definitions for some characters to the definitions of ISO 10646.

UNICODE UNDER LINUX

Under Linux, only the **BMP** at implementation level 1 should be used at the moment, in order to keep the implementation complexity of combining characters low. The higher implementation levels are more suitable for special word processing formats, but not as a generic system character set. The C type **wchar_t** is on Linux an unsigned 16-bit integer type and its values are interpreted as **UCS** level 1 **BMP** codes.

The locale setting specifies, whether the system character encoding is for example **UTF-8** or **ISO 8859-1**. Library functions like **wctomb**, **mbtowc**, or **wprintf** can be used to transform the internal **wchar_t** characters and strings into the system character encoding and back.

PRIVATE AREA

In the **BMP**, the range 0xe000 to 0xf8ff will never be assigned any characters by the standard and is reserved for private usage. For the Linux community, this private area has been subdivided further into the range 0xe000 to 0xefff which can be used individually by any end-user and the Linux zone in the range 0xf000 to 0xf8ff where extensions are coordinated among all Linux users. The registry of the characters assigned to the Linux zone is currently maintained by

H. Peter Anvin <Peter.Anvin@linux.org>, Yggdrasil Computing, Inc. It contains some DEC VT100 graphics characters missing in Unicode, gives direct access to the characters in the console font buffer and contains the characters used by a few advanced scripts like Klingon.

LITERATURE

* Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane. International Standard ISO 10646-1, International Organization for Standardization, Geneva, 1993.

  This is the official specification of **UCS**. Pretty official, pretty thick, and pretty expensive. For ordering information, check www.iso.ch.

* The Unicode Standard - Worldwide Character Encoding Version 1.0. The Unicode Consortium, Addison-Wesley, Reading, MA, 1991.

  There is already Unicode 1.1.4 available. The changes to the 1.0 book are available from ftp.unicode.org. Unicode 2.0 will be published again as a book in 1996.

* S. Harbison, G. Steele. C - A Reference Manual. Fourth edition, Prentice Hall, Englewood Cliffs, 1995, ISBN 0-13-326224-3.

  A good reference book about the C programming language. The fourth edition now covers also the 1994 Amendment 1 to the ISO C standard (ISO/IEC 9899:1990) which adds a large number of new C library functions for handling wide character sets.

BUGS

At the time when this man page was written, the Linux libc support for **UCS** was far from complete.

AUTHOR

Markus Kuhn <mskuhn@cip.informatik.uni-erlangen.de>

SEE ALSO

utf-8(7)

NAME

UTF-8 - an ASCII compatible multibyte Unicode encoding

DESCRIPTION

The **Unicode** character set occupies a 16-bit code space. The most obvious Unicode encoding (known as **UCS-2**) consists of a sequence of 16-bit words. Such strings can contain as parts of many 16-bit characters bytes like '\0' or '/' which have a special meaning in filenames and other C library function parameters. In addition, the majority of UNIX tools expects ASCII files and can't read 16-bit words as characters without major modifications. For these reasons, **UCS-2** is not a suitable external encoding of **Unicode** in filenames, text files, environment variables, etc. The **ISO 10646 Universal Character Set (UCS)**, a superset of Unicode, occupies even a 31-bit code space and the obvious **UCS-4** encoding for it (a sequence of 32-bit words) has the same problems.

The **UTF-8** encoding of **Unicode** and **UCS** does not have these problems and is the way to go for using the **Unicode** character set under Unix-style operating systems.

PROPERTIES

The **UTF-8** encoding has the following nice properties:

* **UCS** characters 0x00000000 to 0x0000007f (the classical **US-ASCII** characters) are encoded simply as bytes 0x00 to 0x7f (ASCII compatibility). This means that files and strings which contain only 7-bit ASCII characters have the same encoding under both **ASCII** and **UTF-8**.

* All **UCS** characters > 0x7f are encoded as a multibyte sequence consisting only of bytes in the range 0x80 to 0xfd, so no ASCII byte can appear as part of another character and there are no problems with e.g. '\0' or '/'.

* The lexicographic sorting order of **UCS-4** strings is preserved.

* All possible 2^31 UCS codes can be encoded using **UTF-8**.

* The bytes 0xfe and 0xff are never used in the **UTF-8** encoding.

* The first byte of a multibyte sequence which represents a single non-ASCII **UCS** character is always in the range 0xc0 to 0xfd and indicates how long this multibyte sequence is. All further bytes in a multibyte sequence are in the range 0x80 to 0xbf. This allows easy resynchronization and makes the encoding stateless and robust against missing bytes.

* **UTF-8** encoded **UCS** characters may be up to six bytes long, however **Unicode** characters can only be up to three bytes long. As Linux uses only the 16-bit **Unicode** subset of **UCS**, under Linux, **UTF-8** multibyte sequences can only be one, two or three bytes long.

ENCODING

The following byte sequences are used to represent a character. The sequence to be used depends on the UCS code number of the character:

0x00000000 - 0x0000007F:
    0*xxxxxxx*

0x00000080 - 0x000007FF:
    110*xxxxx* 10*xxxxxx*

0x00000800 - 0x0000FFFF:
    1110*xxxx* 10*xxxxxx* 10*xxxxxx*

0x00010000 - 0x001FFFFF:

11110*xxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*

0x00200000 - 0x03FFFFFF:
11111*0xx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*

0x04000000 - 0x7FFFFFFF:
1111110**x** 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*

The *xxx* bit positions are filled with the bits of the character code number in binary representation. Only the shortest possible multibyte sequence which can represent the code number of the character can be used.

EXAMPLES

The **Unicode** character 0xa9 = 1010 1001 (the copyright sign) is encoded in UTF-8 as

11000010 10101001 = 0xc2 0xa9

and character 0x2260 = 0010 0010 0110 0000 (the "not equal" symbol) is encoded as:

11100010 10001001 10100000 = 0xe2 0x89 0xa0

STANDARDS

ISO 10646, Unicode 1.1, XPG4, Plan 9.

AUTHOR

Markus Kuhn <mskuhn@cip.informatik.uni-erlangen.de>

SEE ALSO

unicode(7)

Linux Man Pages Section 8

- intro.8
- lilo.8

NAME

intro - Introduction to administration and privileged commands

DESCRIPTION

This chapter describes commands which either can be or are only used by the superuser, like daemons and machine or hardware related commands.

AUTHORS

Look at the header of the manual page for the author(s) and copyright conditions. Note that these can be different from page to page!

TOC

NAME

lilo - install boot loader

SYNOPSIS

Main function:

/sbin/lilo - install boot loader

Auxiliary uses:

/sbin/lilo -q - query map
/sbin/lilo -R - set default command line for next reboot
/sbin/lilo -I - inquire path name of current kernel
/sbin/lilo {-u|-U} - uninstall lilo

DESCRIPTION

lilo installs a boot loader that will be activated next time you boot. It has lots of options.

-v Increase verbosity. Giving one or more **-v** options will make lilo more verbose.

-q List the currently mapped files. **lilo** maintains a file, by default */boot/map*, containing the name and location of the kernel(s) to boot. This option will list the names therein.

-m *map-file*
Use specified map file instead of the default.

-C *config-file*

lilo reads its instructions about what files to map from its config file, by default */etc/lilo.conf*. This option can be used to specify a non-default config file.

-d *delay*
If you have specified several kernels, and press Shift at boot-time, the boot loader will present you with a choice of which system to boot. After a timeout period the first kernel in the list is booted. This option specifies the timeout delay in deciseconds.

-D *label*
Use the kernel with the given label, instead of the first one in the list, as the default kernel to boot.

-r *root-directory*
Before doing anything else, do a *chroot* to the indicated directory. Used for repairing a setup from a boot floppy.

-t Test only. Do not really write a new boot sector or map file. Use together with **-v** to find out what **lilo** is about to do.

-c Enable map compaction. This will merge read requests from adjacent sectors. Speeds up the booting (especially from floppy).

-f *disk-tab*
Specify disk geometry parameter file. (The default is */etc/disktab*.)

-i *boot-sector*
Specify a file to be used as the new boot sector. (The default is */boot/boot.b*.)

-l Generate linear sector addresses instead of sector/head/cylinder addresses.

-P {*fix*|*ignore*}
Fix (or ignore) `corrupt' partition tables, i.e., partition tables with linear and sector/head/cylinder addresses that do not correspond.

-s *save-file*

When **lilo** overwrites the boot sector, it preserves the old contents in a file, by default */boot/boot.NNNN* where NNNN depends on the device. This option specifies an alternate save file for the boot sector. (Or, together with the **-u** option, specifies from where to restore the boot sector.)

-**S** *save-file*
Normally, **lilo** will not overwrite an existing save file. This options says that overwriting is allowed.

-**u** *device-name*
Uninstall **lilo**, by copying the saved boot sector back. A time-stamp is checked.

-**U** *device-name*
Idem, but do not check the time-stamp.

-**R** *command line*
This option sets the default command for the boot loader the next time it executes. The boot loader will then erase this line: this is a once-only command. It is typically used in reboot scripts, just before calling `shutdown -r'.

-**I** *label*
The label of the running kernel can be found in the environment variable BOOT_IMAGE after startup. This command will print the corresponding path name on stdout.

-**V** Print version number.

The above command line options correspond to the key words in the config file indicated below.

l l. -b bootdev boot=bootdev -c compact -d dsec delay=dsec -D label default=label -i bootsector install=bootsector -f file disktab=file -l linear -m mapfile map=mapfile -P fix fix-table -P ignore ignore-table -s file backup=file -S file force-backup=file -v verbose=level

SEE ALSO

lilo.conf(5).
The lilo distribution comes with very extensive documentation.

AUTHOR

Werner Almesberger (almesber@bernina.ethz.ch).

Linux Man Pages Section 9

- MAJOR.9
- MAP_NR.9
- MINOR.9
- MKDEV.9
- MOD_DEC_USE_COUNT.9
- MOD_INC_USE_COUNT.9
- __skb_dequeue.9
- __skb_insert.9
- __skb_queue_head.9
- __skb_queue_tail.9
- __skb_unlink.9
- atomic_add.9
- atomic_dec.9
- atomic_inc.9
- atomic_read.9
- atomic_set.9
- atomic_sub.9
- cli.9
- copy_from_user.9
- copy_to_user.9
- disable_bh.9
- free_irq.9
- get_user.9
- init_bh.9
- init_module.9
- intro.9
- kfree.9
- kmalloc.9
- mark_bh.9
- mem_map_reserve.9
- mem_map_unreserve.9

- pcibios_find_class.9
- pcibios_find_device.9
- pcibios_present.9
- pcibios_read_config_byte.9
- pcibios_read_config_dword.9
- pcibios_read_config_word.9
- pcibios_strerror.9
- pcibios_write_config_byte.9
- pcibios_write_config_dword.9
- pcibios_write_config_word.9
- poll_wait.9
- printk.9
- probe_irq_off.9
- probe_irq_on.9
- proc_dir_entry.9
- proc_net_register.9
- proc_net_unregister.9
- proc_register.9
- proc_register_dynamic.9
- proc_scsi_register.9
- proc_scsi_unregister.9
- proc_unregister.9
- put_user.9
- register_chrdev.9
- register_console.9
- remove_bh.9
- request_irq.9
- save_flags.9
- skb_append.9
- skb_dequeue.9
- skb_insert.9
- skb_peek.9
- skb_queue_empty.9

- skb_queue_head.9
- skb_queue_head_init.9
- skb_queue_len.9
- skb_queue_tail.9
- skb_unlink.9
- skel.9
- sleep_on.9
- sti.9
- wake_up.9

TOC

NAME

MAJOR, MINOR, MKDEV - macros to manipulate device major/minor numbers

SYNOPSIS

```
#include <linux/fs.h>

#define MAJOR(dev)
#define MINOR(dev)
#define MKDEV(major, minor)
```

DESCRIPTION

The **MAJOR** and **MINOR** macros extract from a device id (of type *dev_t)* the major and minor numbers respectively. It is the combined device number that is found in the **struct~file** structure.

The **MKDEV** macro assembles a device number from the supplied major and minor numbers.

RETURN VALUE

Describe the return values. Enumerate all the distinct values and all the ranges.

AVAILABILITY

Linux 0.0+

In later kernels (Linux 2.1 and later) there is movement towards using 64bit device numbers. Although it is generally a good idea to not presume to know how device numbers are constructed, it is during this period of transition fundamentally important that you ONLY use these functions to manipulate device numbers.

SEE ALSO

mknod(1)

Also list some source files for the kernel that implement the functions of the page.

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS

These are macros, so parameters may be evaluated multiple times.

NAME

MAP_NR - get memory map index for page in kernel memory

SYNOPSIS

#include <linux/mm.h>

#define MAP_NR(*page***)**

DESCRIPTION

This macro maps the page of kernel memory to an index into the **mem_map** array. The input page is kernel-space page addresses such as those that come from the **get_free_page**(9) function. The expression *mem_map*[*MAP_NR*(*page*)] returns a pointer to a *mem_map_***t** for the page in question.

Device drivers need access to the page structure if it is implimenting some special kind of memory segment, i.e. cache for a virtual frame buffer.

RETURN VALUE

Returns an index into the **mem_map** array.

AVAILABILITY

Linux 2.0+

SEE ALSO

get_free_pages(9)

/usr/include/asm/page.h

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS

There is no check that the page you are passing really is a valid page address.

TOC

NAME

MOD_INC_USE_COUNT, MOD_DEC_USE_COUNT - Support reference counting of modules

SYNOPSIS

```
#include <linux/module.h>

#define MOD_INC_USE_COUNT
#define MOD_DEC_USE_COUNT
```

DESCRIPTION

These macros support reference counting of modules during their lifetime. Each time **MOD_INC_USE_COUNT**(9) is invoked, the kernel increases the number of counted references to the module. The **MOD_DEC_USE_COUNT**(9) decreases the reference count.

The reference count is used by the kernel to know when a module is no longer in use by the device or application software. A **MOD_INC_USE_COUNT**(9) is generally placed in the open entry for a driver, and a **MOD_DEC_USE_COUNT**(9) in the release for the driver, to count the number of times the device has been opened, and thus prevent the module being unloaded when in use. The device driver may increment the use count other times, such as when memory is mapped or the module must remain loaded for external events.

If the device driver is not compiled as a module is not defined) then the macro reduces to no behavior.

RETURN VALUE

These macros take no parameters and return no result.

AVAILABILITY

Linux 1.0+

SEE ALSO

/usr/include/linux/module.h

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS

NAME

atomic_add, atomic_sub, atomic_inc, atomic_dec - thread/SMP safe arithmetic on atomic data

SYNOPSIS

```
#include <asm/atomic.h>

void atomic_add(int i, volatile atomic_t*v)
void atomic_sub(int i, volatile atomic_t*v)
void atomic_inc(volatile atomic_t*v)
void atomic_dec(volatile atomic_t*v)
int atomic_read(volatile atomic_t*v)
void atomic_set(volatile atomic_t*v, int i)
int atomic_dec_and_test(volatile atomic_t*v)
```

DESCRIPTION

These functions manipulate variables of type **atomic_t** is SMP and interrupt safe ways. These variables can be used to hold spin locks or SMP-safe reference counters. These functions guarantee that the operation that they represent is performed correctly. If necessary, hardware bus locking is performed to protect the operation. Usually, the CPU has some sort of atomic instructions that allow these operations to be performed quickly and safely.

The **atomic_dec_and_test** decrements the atomic variable, and returns true if the result is zero. This function is particularly useful in implementing spin locks on SMP systems.

RETURN VALUE

The **atomic_read** function returns the integer value of the atomic variable.

The **atomic_dec_and_test** returns TRUE if the value becomes zero after the decrement.

AVAILABILITY

Linux 2.0+

SEE ALSO

intro(9)

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS

The read and set operations generally have no special protections.

NAME

cli, sti - disable/enable interrupts

SYNOPSIS

```
#include <asm/system.h>

extern void cli()
extern void sti()
```

DESCRIPTION

The **cli** function causes interrupts to be blocked on the host, so that following critical code may run uninterrupted. The **sti** function enables interrupts again.

Although it is possible to use cli/sti pairs to protect critical code, it is best to use **cli** with the **save_flags** macro. See **save_flags (9).**

RETURN VALUE

none

AVAILABILITY

Linux 1.0+

SEE ALSO

save_flags(9)

include/asm-/system.h*

AUTHOR

Stephen Williams (steve@icarus.com)

BUGS

The i386 ancestry of the names can be slightly confusing.

TOC

NAME

get_user, put_user, copy_from_user, copy_to_user - copy data between kernel space and user space

SYNOPSIS

```
#include <asm/uaccess.h>

err = get_user ( x, addr );
err = put_user ( x, addr );

bytes_left = copy_from_user(void*to, const void *from,
                            unsigned long n );
bytes_left = copy_to_user(void*to, const void *from,
                          unsigned long n );
```

DESCRIPTION

These macros transfer data between kernel space and user space. In the first example, the kernel variable **x** gets the value of the thing pointed to by *addr* (in user space). For **put_user**, the value of the variable **x** is written starting at the address *addr*. Note well that **x** is a **variable**, not a pointer. *addr* must be a properly typed pointer, since its type determines number of bytes that are copied.

copy_from_user copies **n** bytes from address *from* in user space to address *to* in kernel space. **copy_to_user** copies in the opposite direction.

None of these need the old **verify_area**() call, as they do all the area verification on their own using the paging unit in the CPU hardware. Less chance of missing address verifi-

cation that way. Also, the new address verification is much faster than the old scheme was.

RETURN VALUE

get_user and **put_user** return **0** for success and **-EFAULT** for bad access. **copy_from_user** and **copy_to_user** return the number of bytes they **failed** to copy (so again zero is a successful return).

EXAMPLES

```
if (get_user(type, (char *)arg))
return -EFAULT;
switch (type) {
...

b.maxwidth = 768;
b.maxheight = 576;
b.minwidth = 32;
b.minheight = 32;
if(copy_to_user(arg, &b, sizeof(b)))
    return -EFAULT;

return 0;
...
```

AVAILABILITY

Linux 2.1.4+

SEE ALSO

verify_area(9)

AUTHOR

Manual page by Jim Van Zandt <jrv@vanzandt.mv.com>

TOC

NAME

init_bh, remove_bh, mark_bh, disable_bh, enable_bh - split-half interrupt handling

SYNOPSIS

```
#include <linux/interrupt.h>

void init_bh(int nr, void (*routine)(void));
void remove_bh(int nr);
void mark_bh(int nr);
void disable_bh(int nr);
void enable_bh(int nr);
```

DESCRIPTION

Theory

Split-half handling is a means of dividing an interrupt handler into two sections, one of which (known as the top half') is time-critical and one of which (the bottom half') is not.

The top half (the handler registered with **request_irq**(9)) normally moves data between the device and a memory buffer, ensures that the device is in a sane state, and little else. While the top half of a handler is running, the IRQ is question is blocked; if it is a fast interrupt handler (i.e., it has **SA_INTERRUPT** set), all interrupts are disabled.

The bottom half does whatever remains to be done. Bottom halves run with interrupts enabled, although a locking mechanism ensures that only one bottom half will be running at a given time. Bottom halves are run by **do_bottom_half()**,

which is called from **schedule()** and **ret_from_sys_call()**.

Usage

init_bh() installs *routine*() as bottom half number *nr*. It operates by adding an entry to the **bh_base[]** table, and setting the appropriate bit of the **bh_mask** vector. Rather than specifying a number explicitly, one should add an entry to the anonymous enum in *include/linux/interrupt.h*.

remove_bh() removes bottom half number *nr* from the list of bottom halves. It removes the entry from **bh_base[]** and clears the appropriate bit of **bh_mask**.

mark_bh() requests that the kernel run the specified bottom half at the first available opportunity. This function is normally called from the top half of an interrupt handler. It operates by setting the appropriate bit of the **bh_active** vector.

disable_bh() disables bottom half number *nr* by clearing the appropriate bit of **bh_mask**. This function also increments **bh_mask_count[***nr***]**, which is used to ensure that nested calls to **disable_bh()** must be matched by an equal number of calls to **enable_bh()**.

enable_bh() enables a bottom half previously disabled by **disable_bh()**. This function decrements **bh_mask_count[***nr***]**. Then, if that value is zero, the specified bottom half is enabled by setting the appropriate bit of **bh_mask**.

RETURN VALUE

No value is returned.

AVAILABILITY

Linux 2.0+. **init_bh()** and **remove_bh()** were not present in older versions on Linux. Under those versions, **bh_base[]**

and **bh_mask** must be modified by hand.

SEE ALSO

request_irq(9), **queue_task**(9)

include/asm/softirq.h*, *include/linux/interrupt.h*, *kernel/softirq.c*

Kernel Korner in issue 26 of *The Linux Journal* includes a discussion of split-half interrupts under Linux. An online copy of this article can be found at **http://www.ssc.com/lj/issue26/interrupt.html**.

AUTHOR

Neil Moore <*amethyst@maxwell.ml.org*>

BUGS

Only 32 bottom halves are allowed. Increasing this number requires changing the size of **bh_base[]** and **bh_mask_count[]** in *kernel/softirq.c*, and changing **bh_active** and **bh_mask** (in the same file) to a larger type. A better solution, however, would be to consolidate multiple bottom halves into a single one by using task queues. See **queue_task**(9) for details.


TOC

NAME

init_module, cleanup_module - module load and unload functions

SYNOPSIS

```
#include <linux/module.h>
#include <linux/modversions.h>

int init_module(void);
void cleanup_module(void);
```

DESCRIPTION

These functions are not part of the kernel but entry points into loadable modules. These are the only symbols that must be externally defined in order to load a module into a running kernel.

When a module is loaded into a running kernel, the **init_module**(9) function within that object file is called to set up the module. The implementation of that function initializes local features and uses functions such as **register_chrdev**(9) to attach itself to the kernel. It then returns zero(0) if it succeeds. If there is a problem or the module decides that it cannot be loaded, it returns instead an error code (i.e. *-ENODEV)* and the kernel releases the module again.

Once loaded, the **cleanup_module**(9) function of a module is used by the kernel to remove the module again. The module detaches itself from the kernel and returns.

RETURN VALUE

The **init_module**(9) function returns 0 on success, or an error code <0 if the module cannot be initialized.

AVAILABILITY

Linux 1.0+

SEE ALSO

MOD_INC_USE_COUNT(9), **insmod**(1)

Also list some source files for the kernel that implement the functions of the page.

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS

It is entirely up to the driver to be sure it is detached from the kernel when unloaded. If a module makes an error in this regard, bad things may happen.

NAME

intro - Introduction to kernel interface

SYNOPSIS

#include <linux/version.h>

DESCRIPTION

This section documents the functions available to device driver writers and kernel level modules. The functions are of interest mainly to device driver writers, although anyone considering running code in linux kernel mode may need to be familiar with these interfaces.

Some of the functions of the DDI exist only in certain versions of the kernel. Use the **LINUX_VERSION_CODE** macro to test for specific versions of the kernel. For example, to use a feature that is new to 2.1, say:

```
#if LINUX_VERSION_CODE >= 0x020100
    ... use new stuff ...
#else
    ... do it the old way ...
#endif
```

The following is a list of the man pages, divided roughly into function groups.

Kernel Functions
These are general kernel functions.

MAJOR

MOD_INC_USE_COUNT

cli

get_user

init_bh

init_module

kmalloc

poll_wait

printk

probe_irq_on

register_chrdev

register_console

request_irq

save_flags

sleep_on

wake_up

/proc functions

These functions relate to manipulation of the **/proc** filesystem.

proc_dir_entry

proc_net_register

proc_scsi_register

BIOS32 functions

These are specific to PCI (BIOS32) support.

pcibios_find_class

pcibios_present

pcibios_read_config_byte

pcibios_read_config_dword

pcibios_read_config_word

pcibios_strerror

pcibios_write_config_byte

pcibios_write_config_dword

pcibios_write_config_word

VM functions

These are functions that support manipulating the virtual memory subsystem.

MAP_NR

mem_map_reserve

Network Functions

skb_dequeue

skb_insert

skb_peek

skb_queue_empty

skb_queue_head

skb_queue_head_init

skb_queue_len

skb_queue_tail

skb_unlink

AVAILABILITY

Each man page attempts to list the kernel versions where the function is available. If the form of the function changes, this section tells when the described form applies.

SEE ALSO

This section lists other man pages that may be of interest. Also, interesting source files in the linux kernel may be listed here.

AUTHORS

Each man page has a section like this one that lists the author(s) who contributed significantly to that page. Other unnamed individuals may also have contributed corrections, editorial, etc.

Major contributors are (in alphabetical order) Cyrus Durgin <cider@speakeasy.org>, Niel Moore <amethyst@maxwell.ml.org>, Keith Owens <kaos@ocs.com.au>, Kirk Petersen <kirk@speakeasy.org>, Jim Van Zandt <jrv@vanzandt.mv.com>, and Stephen Williams <steve@icarus.com>.

Editorial, and this intro page, were done by Stephen Williams <steve@icarus.com>.

BUGS

The living linux kernel is a moving target, and the kernel functions are unique to linux. Therefore, although the editor and contributers make a good effort to be as accurate as

possible, errors may exist. The source codes of the linux kernel are the ultimate authority on the behavior of any function and should be considered the final word.

NAME

kmalloc, kfree - allocate and free pieces of memory

SYNOPSIS

```
#include <linux/malloc.h>

void * kmalloc (size_t size, int priority));
void kfree (void * __ptr));
```

DESCRIPTION

The **kmalloc** function allocates a piece of memory.

The parameter *size* is the number of bytes that will be allocated. The parameter *priority* indicates the importance and type of the memory request. Some possible values are GFP_DMA, GFP_ATOMIC, GFP_BUFFER, and GFP_NFS.

The **kfree** function releases a piece of memory that is passed as the *__ptr* parameter.

RETURN VALUE

On success, **kmalloc** returns a pointer to the newly allocated memory.

If there is an error, NULL is returned instead.

AVAILABILITY

Linux 2.0

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

NAME

mem_map_reserve, mem_map_unreserve - Manipulate flags of virtual memory pages.

SYNOPSIS

```
#include <linux/mm.h>
#include <linux/wrapper.h>

#define mem_map_reserve(page_nr)
#define mem_map_unreserve(page_nr)
```

DESCRIPTION

These macros cause a page to become reserved/unreserved. A reserved page is reserved from any further consideration by the linux kernel, meaning it is not scanned as potentially pageable, or available for page allocation. The kernel treats reserved pages as memory-mapped hardware.

It makes sense for a driver to mark a page reserved, for example, if the driver supports **mmap**(2) with dynamically allocated pages that the target device can access via DMA.

RETURN VALUE

None

AVAILABILITY

Linux 2.0+

SEE ALSO

MAP_NR(9), **get_free_pages**(9), **mmap**(2)

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS

If a page is marked reserved, a call to **free_page**(9) will silenty ignore it. If the page was originally allocated by get_free_page, you must remember to unmark the page before releasing it to the system. Otherwise, the page is lost.

NAME

pcibios_find_class, pcibios_find_device - find a PCI class or device

SYNOPSIS

#include <linux/bios32.h>

index, unsigned char* bus

int pcibios_find_class(unsigned int *class_code*, **unsigned short**

unsigned short device_id , unsigned short

int pcibios_find_device(unsigned short *vendor*,

DESCRIPTION

The **pcibios_find_class** function searches for a certain class of device on the PCI bus. In this instance, class is used to represent a few different categories of devices. Some examples are storage, network, display, and memory classes. The classes are defined by the PCI Specification, and the *class_code* is matched with the dword in the configuration


TOC

space of the device at offset 0x08.

The **pcibios_find_device** function finds a device on the PCI bus that has the matching device and vendor ids. The vendor ids are assigned to the PCI Sig to vendors, who in turn assign device ids to the devices they develop. The vendor and device ids are encoded in dword 0x00 of the configuration space.

Both **pcibios_find_class** and **pcibios_find_device** include an extra *index* parameter, which is used to select the specific device if there are multiple matches. An *index* of 0 matches the firts located board, 1 the second, and so on.

The *bus* and *device_fn* parameters are PCI specific cookies that are passed to other pcibios functions to access the configuration space of the located device.

RETURN VALUE

On success, the functions return *PCIBIOS_SUCCESSFUL*. Otherwise, one of the following error codes is returned:

PCIBIOS_DEVICE_NOT_FOUND
: There is no device that matches the search criteria,

PCIBIOS_BAD_VENDOR_ID
: The vendor id is invalid. (0xffff is not a valid id.)

PCIBIOS_FUNC_NOT_SUPPRTED
: The PCI subsystem is not available

AVAILABILITY

Linux 1.0+

SEE ALSO

pcibios_present(9)

/usr/include/linux/pci.h, /usr/include/linux/bios32.h

PCI Local Bus Specification

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

TOC

NAME

pcibios_present - determine whether a PCI bus is available

SYNOPSIS

```
#include <linux/bios32.h>

int pcibios_present(void);
```

DESCRIPTION

The **pcibios_present** tests for the presence of PCI support on the local host. It returns true (!0) if PCI support exists, false otherwise.

RETURN VALUE

Zero if PCI support is NOT present.

AVAILABILITY

Linux 1.0+

SEE ALSO

/usr/include/linux/bios32.h,
/usr/src/linux/arch/*/kernel/bios32.c

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

Bugs?! This is way to trivial to have bugs.

TOC

NAME

pcibios_read_config_byte - read one byte of data from the configuration space of the PCI bus

SYNOPSIS

#include <linux/bios32.h>

int pcibios_read_config_byte (unsigned char bus, unsigned char device_fn,unsigned char where, unsigned char * value);

DESCRIPTION

The **pcibios_read_config_byte** function reads one byte from the configuration space of the PCI bus. The bus can be specified with the **bus** parameters. The **device_fn** paramter determines which ? to use. **where** is set to the ?. After the call is made, **value** points to the value that was read.

RETURN VALUE

The return value is taken from the PCI controller in a way that I'm not quite sure of.

If the PCI system is unavailable, PCIBIOS_FUNC_NOT_SUPPORTED is returned.

AVAILABILITY

Linux 1.0?

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

NAME

pcibios_read_config_dword - read a double word of data from the configuration space of the PCI bus

SYNOPSIS

```
#include <linux/bios32.h>

int pcibios_read_config_dword (unsigned char bus, unsigned char device_fn,unsigned char where, unsigned int * value);
```

DESCRIPTION

The **pcibios_read_config_dword** function reads a double word from the configuration space of the PCI bus. The bus can be specified with the **bus** parameters. The **device_fn** paramter determines which ? to use. **where** is set to the ?. After the call is made, **value** points to the value that was read.

RETURN VALUE

The return value is taken from the PCI controller in a way that I'm not quite sure of.

If the PCI system is unavailable, PCIBIOS_FUNC_NOT_SUPPORTED is returned.

AVAILABILITY

Linux 1.0?

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

TOC

NAME

pcibios_read_config_word - read a word of data from the configuration space of the PCI bus

SYNOPSIS

#include <linux/bios32.h>

int pcibios_read_config_word (unsigned char bus, unsigned char device_fn,unsigned char where, unsigned short * value);

DESCRIPTION

The **pcibios_read_config_word** function reads a word from the configuration space of the PCI bus. The bus can be specified with the **bus** parameters. The **device_fn** paramter determines which ? to use. **where** is set to the ?. After the call is made, **value** points to the value that was read.

RETURN VALUE

The return value is taken from the PCI controller in a way that I'm not quite sure of.

If the PCI system is unavailable, PCIBIOS_FUNC_NOT_SUPPORTED is returned.

AVAILABILITY

Linux 1.0?

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

NAME

pcibios_strerror - Convert BIOS32 return codes to strings

SYNOPSIS

#include <linux/bios32.h>

const char* pcibios_strerror(int *error*));

DESCRIPTION

The **pcibios_strerror** function converts a PCI BIOS32 error number into a human readable error message. The bios32 error codes are defined by the PCI standard.

RETURN VALUE

The return value is a pointer to the error message.

SEE ALSO

/usr/include/linux/bios32.h,
/usr/src/linux/arch/*/kernel/bios32.c

PCI Local Bus -- PCI BIOS SPECIFICATION

AVAILABILITY

Linux 1.0?

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

The result may or may not be overwritten by a subsequent call to this function. it is best to assume it does.

NAME

pcibios_write_config_byte - write one byte of data to the configuration space of the PCI bus

SYNOPSIS

```
#include <linux/bios32.h>

int pcibios_write_config_byte(unsigned char bus, unsigned char device_fn,unsigned char where, unsigned char value);
```

DESCRIPTION

The **pcibios_write_config_byte** function writes one byte to the configuration space of the PCI bus. The bus can be specified with the **bus** parameters. The **device_fn** paramter determines which ? to use. **where** is set to the ?. The byte to be written to the configuration space is stored in the **value** parameter.

RETURN VALUE

The return value is taken from the PCI controller in a way that I'm not quite sure of.

If the PCI system is unavailable, PCIBIOS_FUNC_NOT_SUPPORTED is returned.

AVAILABILITY

Linux 1.0?

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

NAME

pcibios_write_config_dword - write one double word of data to the configuration space of the PCI bus

SYNOPSIS

```
#include <linux/bios32.h>

int pcibios_write_config_dword (unsigned char bus, unsigned char device_fn,unsigned char where, unsigned int value);
```

DESCRIPTION

The **pcibios_write_config_dword** function writes one double word to the configuration space of the PCI bus. The bus can be specified with the **bus** parameters. The **device_fn** paramter determines which ? to use. **where** is set to the ?. The word to be written to the configuration space is stored in the **value** parameter.

RETURN VALUE

The return value is taken from the PCI controller in a way that I'm not quite sure of.

If the PCI system is unavailable, PCIBIOS_FUNC_NOT_SUPPORTED is returned.

AVAILABILITY

Linux 1.0?

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

NAME

pcibios_write_config_word - write one word of data to the configuration space of the PCI bus

SYNOPSIS

```
#include <linux/bios32.h>

int pcibios_write_config_word (unsigned char bus, unsigned char device_fn,unsigned char where, unsigned short value);
```

DESCRIPTION

The **pcibios_write_config_word** function writes one word to the configuration space of the PCI bus. The bus can be specified with the **bus** parameters. The **device_fn** paramter determines which ? to use. **where** is set to the ?. The word to be written to the configuration space is stored in the **value** parameter.

RETURN VALUE

The return value is taken from the PCI controller in a way that I'm not quite sure of.

If the PCI system is unavailable, PCIBIOS_FUNC_NOT_SUPPORTED is returned.

AVAILABILITY

Linux 1.0?

AUTHOR

Kirk Petersen (kirk@speakeasy.org)

BUGS

NAME

poll_wait - wait for selectable event to be ready

SYNOPSIS

#include <linux/poll.h>

void poll_wait(structwait_queue**sync*, **poll_table****pt*))

DESCRIPTION

This function is used in support of the poll device driver entry point. The intent is for a device driver to put *sync* into the poll table immediately upon entering the device poll routine, then return a bit mask of events that are currently ready. The kernel looks at the mask of events to see if something it needs is ready, and suspends the process if not.

```
static unsigned int xxpoll(struct file*file, poll_table*pt)
{
    poll_wait(&data_in_sync, pt);
    if (data_in.cnt > 0) return POLLIN | POLLRDNORM;
    else return 0;
}
```

This example shows the basic use of **poll_wait**(9) in a device driver poll function.

AVAILABILITY

Linux kernel 2.1.34+

SEE ALSO

intro(9)

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS

To early to say.

TOC

NAME

printk - print messages to console log

SYNOPSIS

#include <linux/kernel.h>

int printk(const char*fmt*, ...)

DESCRIPTION

Print a formatted message to the kernel console, much like the **printf** function of the stdio library. Normally, the message is written to the physical console device of the computer, although this behavior can be changed with the **register_console** function. Messages are also stored in a message log book.

The generated string may also start with a message priority code, which sets the priority of the message. The priority code strings are of the form <**n**> where n is a number from 0 - 7. The following macros are defined in the <*linux/kernel.h*> header file:

KERN_EMERG
System is unuseable

KERN_ALERT
Action must be taken immediately

KERN_CRIT
Critical conditions

KERN_ERR
Error conditions

KERN_WARNING
Warning conditions

KERN_NOTICE
Normal but significant condition

KERN_INFO
Informational

KERN_DEBUG
Debug-level messages

For example

```
printk(KERN_NOTICE "Hello, world.\n");
```

does the expected thing.

RETURN VALUE

Returns the number of characters written to the log.

AVAILABILITY

Linux 1.0+

SEE ALSO

register_console(9), **syslog**(2)

kernel/printk.c

AUTHOR

Stephen Williams (steve@icarus.com)

BUGS

float and double formats are not supported. Floats and doubles do not belong inside the kernel anyhow.

The **printk** implementation protects itself from interruption, so in principle it can be used in interrupts handlers and critical sections. However, there are no guarantees about the console function that is registered.

TOC

TOC

NAME

probe_irq_on, probe_irq_off - safe probing for IRQs

SYNOPSIS

```
#include <linux/interrupt.h>

unsigned long probe_irq_on(void)
int probe_irq_off(unsigned long irqs));
```

DESCRIPTION

Usage

probe_irq_on() turns on IRQ detection. It operates by enabling all interrupts which have no handlers, while keeping the handlers for those interrupts NULL. The kernel's generic interrupt handling routine will disable these IRQs when an interrupt is received on them. **probe_irq_on()** adds each of these IRQ numbers to a vector which it will return. It waits approximately 100ms for any spurious interrupts that may occur, and masks these from its vector; it then returns this vector to its caller.

probe_irq_off() tests an internal list of enabled IRQs against its *irqs* parameter, which should be the value returned by the last **probe_irq_on()**. This function basically detects which IRQs have been switched off, and thus which ones have received interrupts.

Example

This explanation may seem a bit confusing, so here is an example of code the mythical FUBAR 2000 driver could use to probe for IRQs:


```
unsigned long irqs;
int irq;

irqs = probe_irq_on();

outb(FB2K_GIVE_ME_AN_INTERRUPT_OR_GIVE_ME_DEATH,
     FB2K_CONTROL_PORT);
  /* the interrupt could take a while to occur */
udelay(1000);

irq = probe_irq_off(irqs);

if (irq == 0) {
     printk("fb2k: could not detect IRQ.\n");
     printk("fb2k: Installation failed.\n");
} else if (irq == -1) {
     printk("fb2k: multiple IRQs detected.\n");
     printk("fb2k: Installation failed.\n");
} else {
     fb2k_dev->irq = irq;
     printk("fb2k: using probed IRQ %d.\n", irq);
}
```

RETURN VALUE

probe_irq_on() returns a bitmap of all unhandled IRQs (except those which are receiving spurious interrupts). This value should only be used as a parameter to the next call to **probe_irq_off()**.

probe_irq_off() returns the IRQ number of whichever unhandled interrupt has occurred since the last **probe_irq_on()**. If no interrupts have occurred on any of the marked IRQs, 0 is returned; if interrupts have occurred on more than one of these IRQs, -1 is returned.

AVAILABILITY

Linux 1.2+. These functions are not available on m68k-based machines.

SEE ALSO

request_irq(9)

arch//kernel/irq.c*

AUTHOR

Neil Moore <*amethyst@maxwell.ml.org*>

BUGS

As mentioned above, these functions are not available on m68k-based machines.

This manpage is way too confusing.

TOC

TOC

NAME

proc_dir_entry, proc_register, proc_register_dynamic, proc_unregister - register entries in the /proc filesystem.

SYNOPSIS

#include <linux/proc_fs.h>

struct proc_dir_entry * child);

int proc_register(struct proc_dir_entry * *parent*,

int proc_unregister(struct proc_dir_entry * *parent*, **int** *inode*));

struct proc_dir_entry * child);

int proc_register_dynamic(struct proc_dir_entry * *parent*,

DESCRIPTION

The **proc_register** functions add file or directory entries to the /proc file system. They associate processing routines with each node of the /proc tree. The structure **proc_dir_entry** is defined as


```
struct proc_dir_entry {
        unsigned short low_ino;
        unsigned short namelen;
        const char *name;
        mode_t mode;
        nlink_t nlink;
        uid_t uid;
        gid_t gid;
        unsigned long size;
        struct inode_operations * ops;
        int (*get_info)(char *buffer, char **start,
                off_t offset, int length, int unused);
        void (*fill_inode)(struct inode *);
        struct proc_dir_entry *next, *parent, *subdir;
        void *data;
};
```

low_ino The inode number of this directory entry. For **proc_register** this number should be unique within the /proc filesystem, values are defined in <*linux/proc_fs*.**h**>. For **proc_register_dynamic** the inode number is dynamically assigned.

namelen The length of the name, excluding the trailing null.

name The name of this node.

mode The node's type and permissions. Drawn from <*linux/stat.h*>.

nlink Number of links to the node. Initialise to 2 if mode includes S_IFDIR, 1 otherwise.

uid The uid that owns the node, normally 0.

gid The gid that owns the node. normally 0.

size Sets the size of the node, the value will appear as the inode size in listings and be returned by **stat**. Unless you really need a size, set this to zero.

ops Defines the set of inode operations to perform for

your /proc node. For a directory node, use *&proc_dir_inode_operations* unless you have special requirements. For a leaf node, set to NULL unless you have special requirements.

get_info
: If defined, this proc is called when the node is read. Should be NULL for directory nodes. **NOTE:** If you need to return large amounts of data, the proc must return the data in chunks and reposition itself on the next call, using the *offset* variable. See *ip_masq_procinfo* for example code with large output.

fill_inode
: Dynamically fill in the inode characteristics during directory operations. Not normally required and set to NULL. See proc_pid_fill_inode for example code.

next, parent, subdir
: Maintained by /proc routines. Initial value is irrelevant, set to NULL.

data
: An opaque pointer which can be used by proc handlers to pass local data around. Set to whatever you like when calling **proc_register**, normally NULL. This pointer is copied into the inode u.ip_generic field (by proc_get_inode) so it is available to any proc routines that are passed an inode.

proc_register adds the **child** as a node under the **parent**.

proc_register_dynamic dynamically assigns an inode number then adds the **child** as a node under the **parent**.

proc_unregister scans the inode list under the **parent** for the specified **inode** number and removes the matching entry.

RETURN VALUE

proc_register always returns 0.

proc_register_dynamic returns 0 for success or **-EAGAIN** if

there are no free dynamic inode numbers.

proc_unregister returns 0 for success or **-EINVAL** if the node was not found.

SEE ALSO

proc_net_register(9), **proc_net_unregister**(9), **proc_scsi_register**(9),

AUTHOR

Keith Owens <kaos@ocs.com.au>

BUGS

The uniqueness of /proc inode numbers is assumed, not enforced. It is possible to add two nodes with the same inode number.

TOC

NAME

proc_net_register, proc_net_unregister - register network entries in the /proc filesystem

SYNOPSIS

```
#include <linux/proc_fs.h>

int proc_net_register(struct proc_dir_entry * child));

int proc_net_unregister(int inode));
```

DESCRIPTION

These are wrapper functions around **proc_register** and **proc_unregister**. They always use a parent of proc_net.

RETURN VALUE

As for **proc_register** and **proc_unregister**.

SEE ALSO

proc_register(9), **proc_unregister**(9)

AUTHOR

Keith Owens <kaos@ocs.com.au>

NAME

proc_scsi_register, proc_scsi_unregister - register SCSI entries in the /proc filesystem

SYNOPSIS

#include <linux/proc_fs.h>

struct proc_dir_entry * child);

int proc_scsi_register(struct proc_dir_entry * *driver*,

int inode);

int proc_scsi_unregister(struct proc_dir_entry * *driver*,

DESCRIPTION

These are wrapper functions around **proc_register** and **proc_unregister**.

proc_scsi_register always sets the *ops* field of **child** to proc_scsi_inode_operations. If the child inode number is less than PROC_SCSI_FILE, the child is registered with a

parent of proc_scsi and **driver** is ignored. Otherwise the child is registered with a parent of **driver**.

proc_scsi_unregister. If the child inode number is less than PROC_SCSI_FILE, the child is unregistered with a parent of proc_scsi and **driver** is ignored. Otherwise the child is unregistered with a parent of **driver** and scsi_init_free is called on the child.

RETURN VALUE

As for **proc_register** and **proc_unregister**.

SEE ALSO

proc_register(9), **proc_unregister**(9)

AUTHOR

Keith Owens <kaos@ocs.com.au>

NAME

register_chrdev, unregister_chrdev - register a driver major number

SYNOPSIS

```
#include <linux/fs.h>

int register_chrdev(unsigned int major, const char*name,
struct file_operations*ops));
int unregister_chrdev(unsigned int major, const char *name));
```

DESCRIPTION

The **register_chrdev** function associates a character major number with set of driver entry points. The file_operations structure contains pointers to functions that the the driver uses to implement the kernel interface to the driver.

The paramter *major* is the character major number assigned to the device driver and to be mapped to the function table. The *name* parameter is a short name for the device and is displayed in the The */proc/devices* list. It also must exactly match the name passed to **unregister_chrdev** function when releasing the functions.

A device driver module may register as many different major numbers as it supports, though this is not typically done.

The **unregister_chrdev** function releases the major number, and is normally called in the module_cleanup function to remove the driver from the kernel.

RETURN VALUE

On success, **register_chrdev** returns 0 if *major* is a number other then 0, otherwise Linux will choose a major number and return the chosen value.

If there is an error, one of the following codes is returned instead:

-EINVAL
The specified number is not valid (> MAX_CHRDEV)

-EBUSY
The major number is busy

The **unregister_chrdev** function will return 0 if successful, or **-EINVAL** if the major number is not registered with the matching name.

AVAILABILITY

Linux 1.0+

SEE ALSO

register_blkdev(9) **mknod**(2)

AUTHOR

Stephen Williams (steve@icarus.com)

TOC

BUGS


TOC

NAME

register_console - take on kernel console duties

SYNOPSIS

extern void register_console(void(*proc*)(const **char***))**

DESCRIPTION

This function registers with the kernel a function to display all the kernel messages printed by the **printk**(9) function.

This function is normally called once at boot time by the device driver for the console device, whatever that turns out to be for the machine. The console procedure is only called to print messages that are to be printed, the filtering for log level is done already.

RETURN VALUE

None.

AVAILABILITY

Linux 1.0+

SEE ALSO

printk(9)

/usr/src/linux/kernel/printk.c

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS

Only one console can exist at a time, and there is no way to know if any other device has registered as a console.

TOC

NAME

request_irq, free_irq - register an interrupt handler

SYNOPSIS

```
#include <asm/irq.h>
#include <linux/signal.h>
#include <linux/sched.h>
#include <linux/interrupt.h>

int request_irq(unsigned int irq,
     void (*handler)(int, void *, struct pt_regs *),
     unsigned long irqflags, const char *devname,
     void *dev_id));
void free_irq(unsigned int irq, void *dev_id));
```

DESCRIPTION

Usage

The **request_irq()** function requests that a specified function (the handler) be called whenever the kernel receives a given interrupt. The handler may in turn register a bottom half, which is usually a slower part of the handler which does not need to be executed as soon as the interrupt is received. See **init_bh**(9) for more information on bottom halves.

The *irq* parameter is the interrupt number you want to handle. It must be less than **NR_IRQS** (16 on Intel systems), and there may be additional limitations on the value. See *arch/*/kernel/irq.c* (*intr.c* on m68k machines) for more information.

handler is a pointer to the a pointer to the function that will handle the interrupt. The handler is passed the following parameters:

int *irq*
The interrupt number. By testing the value of this parameter, it is possible for a single function to handle multiple IRQs.

void ***dev_id**
The device ID of this handler (see below).

struct pt_regs ***regs**
The registers stored on the stack of the process that was interrupted. Normally, one shouldn't mess with these, although they can be read to determine, for example, whether the interrupted process was in kernel or user mode.

irqflags is, as the name suggests, a bitmask of flags pertaining to this interrupt handler. Legal bits are:

SA_INTERRUPT
This bit indicates that you are registering a fast interrupt handler. The semantics of this are discussed below.

SA_SHIRQ
This bit indicates that your handler supports sharing an IRQ with other handlers (see also **dev_id* below).

SA_SAMPLE_RANDOM
This bit indicates that this IRQ may be used as an entropy source for */dev/random* and */dev/urandom* (see *drivers/char/random.c)*.

SA_PROBE
This bit indicates that the IRQ is being probed and that the handler being installed is not a real one. It was intended that this value be used internally by **probe_irq_on()** (q.v.), but it is no longer used in 2.1.x kernels. In fact, even with 2.0.x kernels, it is only used by the MIPS architecture. You should not be using this value unless you know what you are doing.

SA_STATIC_ALLOC
(Sparc/Sparc64 only) This bit requests that your **struct irqaction** (see below) be added to a statically allocated array of four handlers, rather than the normal **irq_action[]** table. This is used for IRQs that must be requested early in the boot process, before **kmalloc_init()** has been called.

The *devname* parameter is a short name for the device and is displayed in the */proc/interrupts* list.

The *dev_id* parameter is the device ID. This parameter is usually set to NULL, but should be non-null if you wish to do IRQ sharing. This doesn't matter when hooking the interrupt, but is required so that, when **free_irq()** is called, the correct driver is unhooked. Since this is a **void ***, it can point to anything (such as a device-specific structure, or even empty space), but make sure you pass the same pointer to **free_irq()**.

The **free_irq()** function releases an interrupt handler from operation. It takes as parameters the IRQ to unregister, and the device ID of the handler to be unregistered. You should pass the same values here as you did to **request_irq()**. You probably shouldn't unregister other people's interrupt handlers unless you know what you are doing.

Operation

For most architectures, **request_irq()** operates by allocating memory for a **struct irqaction**, filling out the fields thereof, and adding it to the **irq_action[]** table. **enable_irq()** is then called, which simply tells the kernel to start delivering interrupts to the installed handler. This process is vastly different on m68k machines, where it varies depending on what type of machine (Amiga, Atari, etc.) one is using. **free_irq()** simply removes the entries that **request_irq()** added. Note that some of these names differ depending on the architecture (for example, **struct irqaction** is known as **struct irq_action** on the Power PC). If you need to know more about the internal workings of these functions, you are best off reading the source, as some of this information may have changed by the time you read this (if so, tell me, so I can try to update this page).

Fast Interrupt Handlers

A `fast' interrupt handler (one with **SA_INTERRUPT** set) has the following differences from normal `slow' interrupt handlers:

On the ix86 and MIPS, the handler is called with interrupts disabled (they are enabled by default on these machines; on other machines, they are disabled by default).

On the MIPS, a faster return is used.

On the Alpha, MIPS, Sparc, and Sparc64, a fast and a slow handler may not share the same IRQ.

On all architectures except the m68k and the ix86, a `+' is displayed between the interrupt count and the device name in */proc/interrupts*.

The slow-versus-fast interrupt distinction is slowly being phased out. For example, under 2.0.x on the ix86, **SA_INTERRUPT** enabled a fast return as it still does on the MIPS; this distiction was removed in 2.1.x.

RETURN VALUE

On success, **request_irq()** returns 0 if everything goes as planned. Your interrupt handler will start receiving its interrupts immediately. On failure, **request_irq()** returns:

-EINVAL
The IRQ number you requested was either invalid or reserved, or your passed a NULL pointer for the *handler*() parameter.

-ENOMEM
request_irq() could not allocate enough memory for something (probably the **struct irqaction**).

-EBUSY

The IRQ you requested is already being handled, and the IRQ cannot be shared. This can occur because either the handler being registered or the handler already present does not have **SA_SHIRQ** in its *irqflags* field. In addition, on most architectures, all handlers sharing a single IRQ must be of the same speed; i.e., either all or none of them may have the **SA_INTERRUPT** flag set. Finally, it is possible that your architecture may not support sharing of the IRQ you are attempting to use.

-ENXIO
The m68k returns this value for an invalid IRQ number.

free_irq() does not return a value.

AVAILABILITY

Linux 2.1+. The information on this page should work for 2.0.x, but there may be subtle differences (for example, the semantics of **SA_INTERRUPT** on Intel-based machines). Versions earlier than 2.0 had these functions, but the *dev_id* parameter was missing. If you want your code to work with versions both earlier and later than 2.0, you should protect your code with preprocessor macros using **LINUX_VERSION_CODE**.

SEE ALSO

init_bh(9), **probe_irq_on**(9), *arch/*/kernel/irq.c*, *arch/*/kernel/entry.S*, *include/linux/interrupt.h*, *include/asm*/signal.h*.

AUTHOR

Neil Moore <*amethyst@maxwell.ml.org*>

BUGS

It's not exactly a bug, but **request_irq()** on the m68k is *very* strange compared to the same function on the other supported architectures. You should really read *arch/m68k/kernel/ints.c*, *arch/m68k/atari/ataints.c*, *arch/m68k/amiga/amiints.c*, and *arch/m68k/amiga/cia.c* if you plan on writing drivers for any of these systems.

TOC

NAME

save_flags, restore_flags - save/restore processor state and irq mask

SYNOPSIS

```
#include <asm/system.h>

unsigned long flags;
void save_flags(flags)
void restore_flags(flags)
```

DESCRIPTION

The **save_flags** and **restore_flags** macros cooperate with the **cli** function to provide interrupt protection to critical sections of code. The **save_flags** function saves the current processor state, most specifically the interrupt priority level, in the *flags* value (which must be an l-value). The flags must be passed to a subsequent call to **restore_flags** to restore the processor state.

The **save_flags** macro does not affect the processor state, but drivers may use the **cli**(9) and **sti** functions to disable and enable interrupts after saving the current flags. When the critical section of code is passed, the **restore_flags** returns the processor to its state at the point where the matching **save_flags** was called.

Only values returned by **save_flags** can be passed to **restore_flags.** Any other values may cause unpredictable results, and are certainly not portable.

RETURN VALUE

The **restore_flags** macro saves the processor state in the *flags* parameter, which must be a non-const l-value.

AVAILABILITY

Linux 1.0+

SEE ALSO

cli(9)

AUTHOR

Stephen Williams (steve@icarus.com)

BUGS

How does all this react with symmetric multiprocessor machines?

TOC

NAME

__skb_dequeue, skb_dequeue - remove an sk_buff from the head of a list

SYNOPSIS

```
#include <linux/skbuff.h>

struct sk_buff *__skb_dequeue(struct sk_buff_head *list));

struct sk_buff *skb_dequeue(struct sk_buff_head *list));
```

DESCRIPTION

The **skb_dequeue** function removes the head element of an sk_buff_head. It decrements the list qlen pointer, and cleanly detaches the head from the queue. This function should be used instead of performing this task manually, as it provides a clean, standardized way of manipulating an sk_buff_head, and provides interrupt disabling (see **NOTES** below.)

RETURN VALUE

Returns a pointer to the head element of *list*, or **NULL** if *list* is empty.

NOTES

It is important to note the difference between not only **__skb_dequeue** and **skb_dequeue**, but all the **__skb_** functions and their **skb_** counterparts. Essentially, the **__skb_** functions are non-atomic, and should only be used with interrupts disabled. As a convenience, the **skb_** functions are provided, which perform interrupt disable / enable wrapper functionality in addition to performing their specific tasks.

AVAILABILITY

Linux 1.0+

SEE ALSO

intro(9), **skb_unlink**(9), **skb_insert**(9),

/usr/src/linux/net/netlink.c
/usr/src/linux/net/ax25/af_ax25.c
/usr/src/linux/net/core/datagram.c
/usr/src/linux/net/ipv4/ipmr.c /usr/src/linux/net/ipv4/tcp.c

AUTHOR

Cyrus Durgin <cider@speakeasy.org>

TOC

NAME

__skb_insert, skb_insert, skb_append - insert an sk_buff into a list

SYNOPSIS

```
#include <linux/skbuff.h>

void __skb_insert(struct sk_buff *newsk, struct sk_buff
     *prev, struct sk_buff *next

void skb_insert(struct sk_buff *old, struct sk_buff *newsk))

void skb_append(struct sk_buff *old, struct sk_buff *newsk))
```

DESCRIPTION

skb_insert and **skb_append** are essentially wrapper functions for **__skb_insert** (see **NOTES**, below.) **__skb_insert** inserts *newsk* into *list*, and resets the appropriate next and prev pointers. *prev* and *next* are used to frame *newsk* in *list*. After setting the next and prev pointers in *newsk*, **__skb_insert** sets the prev pointer in *next* and the next pointer in *prev*, sets the list pointer in *newsk*, and increments the qlen counter in *list*.

skb_insert and **skb_append** should be used to add sk_buffs to a list rather than performing this task manually; in addition to performing this task in a standardized way, these functions also provide for interrupt diasabling and prevent list mangling. Both of these functions use the list pointer in *old* to determine to which list *newsk* should be attached.

The **skb_insert** function adds *newsk* to the list before *old*.

The **skb_append** function adds *newsk* to the list after *old*.

RETURN VALUE

None.

NOTES

It is important to note the difference between not only **skb_insert**, **skb_append** and **__skb_insert**, but all the **__skb_** functions and their **skb_** counterparts. Essentially, the **__skb_** functions are non-atomic, and should only be used with interrupts disabled. As a convenience, the **skb_** functions are provided, which perform interrupt disable / enable wrapper functionality in addition to performing their specific tasks.

AVAILABILITY

Linux 1.0+

SEE ALSO

intro(9), **skb_queue_head**(9), **skb_queue_tail**(9)

/usr/src/linux/net/ax25/af_ax25.c
/usr/src/linux/net/core/skbuff.c
/usr/src/linux/net/ipv4/tcp_input.c

/usr/src/linux/net/netrom/nr_in.c

AUTHOR

Cyrus Durgin <cider@speakeasy.org>

TOC

NAME

skb_peek - peek an sk_buff

SYNOPSIS

```
#include <linux/skbuff.h>

struct sk_buff *skb_peek(struct sk_buff_head *list_);
```

DESCRIPTION

The **skb_peek** function extracts the head element of *list_*, without modifying *list_*. It is important to note that this is not necessarily a safe operation, as *list_* maintains the original sk_buff, and other operations on *list_* may effect it. To be safe, either disable interrupts using **cli**(9), call **skb_peek**, copy the data necessary to continue, and re-enable interrupts using **sti**(9), or use **skb_dequeue**(9).

RETURN VALUE

Returns a pointer to sk_buff if there is a next element on the *list_*. Otherwise, returns NULL.

AVAILABILITY

Linux 1.0+

SEE ALSO

intro(9), **skb_unlink**(9), **skb_dequeue**(9)

/usr/src/linux/net/appletalk/ddp.c
/usr/src/linux/net/core/datagram.c
/usr/src/linux/net/ipv4/tcp.c /usr/src/linux/net/ipv4/udp.c
/usr/src/linux/net/unix/af_unix.c

AUTHOR

Cyrus Durgin <cider@speakeasy.org>

NAME

skb_queue_empty - detect an empty skbuff queue

SYNOPSIS

#include <linux/skbuff.h>

int skb_queue_empty(struct sk_buff_head **list*));

DESCRIPTION

The **skb_queue_empty** function checks an skbuff queue for "emptiness". This function provides a quick and easy way to determine if there are any sk_buff elements on a given queue.

RETURN VALUE

If there are any sk_buff elements on the list, then 0 is returned. Otherwise, 1 is returned to indicate that the list is empty.

AVAILABILITY

Linux 1.0+

SEE ALSO

intro(9), **skb_queue_len**(9)

/usr/src/linux/net/core/datagram.c
/usr/src/linux/net/core/dev.c /usr/src/linux/net/core/sock.c
/usr/src/linux/net/ipv4/tcp.c

AUTHOR

Cyrus Durgin <cider@speakeasy.org>

NAME

skb_queue_head - insert an sk_buff at the head of a list

SYNOPSIS

```
#include <linux/skbuff.h>

void  __skb_queue_head(struct sk_buff_head *list, struct
     sk_buff *newsk));
```

*newsk);

```
void  skb_queue_head(struct sk_buff_head *list, struct sk_buff
```

DESCRIPTION

The **skb_queue_head** function adds *newsk* to the head of *list*. Specifically, it sets the list, next, and prev pointers in *newsk*, and increments the qlen element of *list*. It also rearranges the next and prev pointers in the existing head of *list* before returning.

RETURN VALUE

None.

NOTES

It is important to note the difference between not only **__skb_queue_head** and **skb_queue_head**, but all the **__skb_** functions and their **skb_** counterparts. Essentially, the **__skb_** functions are non-atomic, and should only be used with interrupts disabled. As a convenience, the **skb_** functions are provided, which perform interrupt disable / enable wrapper functionality in addition to performing their specific tasks.

AVAILABILITY

Linux 1.0+

SEE ALSO

intro(9), **skb_queue_tail**(9), **skb_insert**(9)

/usr/src/linux/net/ax25/af_ax25.c
/usr/src/linux/net/core/skbuff.c
/usr/src/linux/net/ipv4/tcp_input.c
/usr/src/linux/net/netrom/nr_in.c

AUTHOR

TOC

Cyrus Durgin <cider@speakeasy.org>

TOC

NAME

skb_queue_head_init - prepare an sk_buff_head for use

SYNOPSIS

```
#include <linux/skbuff.h>

void  skb_queue_head_init(struct sk_buff_head *list));
```

DESCRIPTION

The **skb_queue_head_init** function prepares an sk_buff_head for use by the other **skb_** functions, and should be called after declaring and allocating memory for any sk_buff_head elements. Specifically, **skb_queue_head_init** sets the prev and next pointers of the *list* to point back at the *list* itself, and sets the *list* length to 0. The use of sk_buff_head elements which have not been initialized with this function is undocumented and may produce irregular results.

RETURN VALUE

None.

AVAILABILITY

Linux 1.0+

SEE ALSO

intro(9)

/usr/src/linux/net/netlink.c
/usr/src/linux/net/appletalk/aarp.c
/usr/src/linux/net/ipv4/af_inet.c
/usr/src/linux/net/ipx/af_ipx.c
/usr/src/linux/net/unix/af_unix.c

AUTHOR

Cyrus Durgin <cider@speakeasy.org>

NAME

skb_queue_len - determine the length of an sk_buff_head

SYNOPSIS

```
#include <linux/skbuff.h>

__u32 skb_queue_len(struct sk_buff_head *list));
```

DESCRIPTION

The **skb_queue_len** function determines the number of sk_buffs assigned to an sk_buff_head.

RETURN VALUE

skb_queue_len returns the number of sk_buff elements attached to an sk_buff_head. If the sk_buff_head is empty, then 0 is returned.

AVAILABILITY

Linux 1.0+

SEE ALSO

intro(9), **skb_queue_head_init**(9), **skb_queue_head**(9),

/usr/src/linux/net/core/datagram.c

AUTHOR

Cyrus Durgin <cider@speakeasy.org>

NAME

__skb_queue_tail, skb_queue_tail - insert an sk_buff at the tail of a list

SYNOPSIS

```
#include <linux/skbuff.h>

void  __skb_queue_tail(struct sk_buff_head *list, struct
     sk_buff *newsk));
```

*newsk);

```
void  skb_queue_tail(struct sk_buff_head *list, struct sk_buff
```

DESCRIPTION

The **skb_queue_tail** function adds *newsk* to the tail of *list*. Specifically, it sets the list, next, and prev pointers in *newsk*, and increments the qlen element of *list*.

RETURN VALUE

None.

NOTES

It is important to note the difference between not only **__skb_queue_tail** and **skb_queue_tail**, but all the **__skb_** functions and their **skb_** counterparts. Essentially, the **__skb_** functions are non-atomic, and should only be used with interrupts disabled. As a convenience, the **skb_** functions are provided, which perform interrupt disable / enable wrapper functionality in addition to performing their specific tasks.

AVAILABILITY

Linux 1.0+

SEE ALSO

intro(9), **skb_queue_head**(9), **skb_insert**(9)

/usr/src/linux/net/netlink.c
/usr/src/linux/net/appletalk/aarp.c
/usr/src/linux/net/core/skbuff.c
/usr/src/linux/net/ipv4/arp.c
/usr/src/linux/include/net/sock.h

AUTHOR

Cyrus Durgin <cider@speakeasy.org>

TOC

TOC

NAME

__skb_unlink, skb_unlink - remove an sk_buff from its list

SYNOPSIS

```
#include <linux/skbuff.h>

void __skb_unlink(struct sk_buff *skb, struct sk_buff_head
     *list));

void skb_unlink(struct sk_buff *skb));
```

DESCRIPTION

The **skb_unlink** function is a wrapper for **__skb_unlink**. **__skb_unlink** removes *skb* from its sk_buff_head. It decrements the list qlen pointer, and cleanly detaches the sk_buff from its queue. This function should always be used instead of performing this task manually, as it provides a clean, standardized way of manipulating an sk_buff_head, and provides interrupt disabling (see **NOTES** below.) Most users will not call **__skb_unlink** directly, as it requires that two arguments be supplied and does not provide any interrupt handling. **skb_unlink** determines the list from which *skb* should be detached, and disables interrupts.

RETURN VALUE

None.

NOTES

It is important to note the difference between not only **__skb_unlink** and **skb_unlink**, but all the **__skb_** functions and their **skb_** counterparts. Essentially, the **__skb_** functions are non-atomic, and should only be used with interrupts disabled. As a convenience, the **skb_** functions are provided, which perform interrupt disable / enable wrapper functionality in addition to performing their specific tasks.

AVAILABILITY

Linux 1.0+

SEE ALSO

intro(9), **skb_dequeue**(9), **skb_insert**(9),

/usr/src/linux/net/core/skbuff.c
/usr/src/linux/net/ipv4/af_inet.c
/usr/src/linux/net/ipv4/ip_output.c
/usr/src/linux/net/ipv4/tcp.c

AUTHOR

TOC

Cyrus Durgin <cider@speakeasy.org>

NAME

skel - skeleton man page for section 9 entries

SYNOPSIS

#include <linux/linux.h>

DESCRIPTION

Describe the function(s) and its parameters. This section should not be considered an introduction to kernel programming, just an english text description of the function at hand. It is OK to presume some basic knowledge of driver programming.

RETURN VALUE

Describe the return values. Enumerate all the distinct values and all the ranges.

AVAILABILITY

List kernel versions, and if restricted to certain architectures, say so.

SEE ALSO

man(1), **man**(7), **intro**(9)

Also list some source files for the kernel that implement the functions of the page.

AUTHOR

Who are you?

BUGS

Describe any misfeatures or suprises that the use of these functions may lead to. The may not be errors, just unfortunate side effects.

TOC

NAME

sleep_on - synchronization using a condition variable

SYNOPSIS

#include <linux/sched.h>

void sleep_on(struct wait_queue**condition)**)

DESCRIPTION

The **sleep_on** function provides a means for synchronizing between processes and with interrupt handlers. The *condition* parameter is a pointer to a pointer to the opaque wait_queue type. Initialize the condition variable to zero, then pass a pointer to it to the **sleep_on** function. The basic idea is as follows:

```
struct wait_queue*con = 0;
     [...]
sleep_on(&con);
```

While a process is sleeping, it is fully interruptible, no matter what the cpu state when entering the function. The cpu state is restored on being awakened.

If a null(0) is passed to **sleep_on**, it returns immediately, without sleeping. This is a no-op.

RETURN VALUE

The **sleep_on** function only returns when explicitly awakened.

AVAILABILITY

Linux 1+

SEE ALSO

wake_up(9)

/usr/src/linux/kernel/sched.c

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS

A call to *sleep_on*(**0**) seems like an interesting way to test for and momentarily allow interrupts, but that is not what happens.

The **sleep_on** function cannot be called by interrupt handlers.

The function is not atomic with the condition tests that the driver writer might include, so the code executing the

sleep_on function is protected from interrupts. Failure to do so generally leads to race

TOC

TOC

NAME

wake_up - wake up sleeping processes

SYNOPSIS

#include <linux/sched.h>

void wake_up(struct wait_queue**condition)**)

DESCRIPTION

The **wake_up** function is the opposite of the **sleep_on**(9) function in that it awakens processes that have gone to sleep using the same condition variable. All the processes sleeping on the given condition are awakened. If there are no processes sleeping on the condition, then none are affected.

Unlike the **sleep_on**(9) function, **wake_up** does not block and may be called by interrupt handlers. It is in fact the principle means of synchronizing with interrupt events.

If the *condition* parameter is NULL, or there are no processes sleeping on *condition*, the call to **wake_up** is a no-op.

RETURN VALUE

None.

AVAILABILITY

Linux 1+

SEE ALSO

sleep_on(9)

/usr/src/linux/kernel/sched.c

AUTHOR

Stephen Williams <steve@icarus.com>

BUGS