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.

For non-locking counterparts, see **unlocked_stdio**(3).

ERRORS
These functions should not fail and do not set the external variable *errno*. (However, in case **fileno**() detects that its argument is not a valid stream, it must return −1 and set *errno* to **EBADF**.)

CONFORMING TO
The functions **clearerr**(), **feof**(), and **ferror**() conform to C89 and C99.

SEE ALSO
open(2), **fdopen**(3), **stdio**(3), **unlocked_stdio**(3)

NAME
fflush − flush a stream

SYNOPSIS
#include <stdio.h>

int fflush(FILE ***_stream_);**

DESCRIPTION
For output streams, **fflush**() forces a write of all user-space buffered data for the given output or update *stream* via the stream's underlying write function.

For input streams associated with seekable files (e.g., disk files, but not pipes or terminals), **fflush**() discards any buffered data that has been fetched from the underlying file, but has not been consumed by the application.

The open status of the stream is unaffected.

If the *stream* argument is NULL, **fflush**() flushes *all* open output streams.

For a nonlocking counterpart, see **unlocked_stdio**(3).

RETURN VALUE
Upon successful completion 0 is returned. Otherwise, **EOF** is returned and *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 **write**(2).

SEE ALSO
fsync(2), **sync**(2), **write**(2), **fclose**(3), **fileno**(3), **fopen**(3), **setbuf**(3), **unlocked_stdio**(3)

NAME

fopen, fdopen, fileno – stream open functions

SYNOPSIS

#include <stdio.h>

FILE *fopen(const char ***_path_, const char ***_mode_**);**
FILE *fdopen(int _fildes_**, const char ***_mode_**);**
int fileno(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 appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

a+ Open for reading and appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

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, and will be closed when the stream created by **fdopen** is closed. The result of applying **fdopen** to a shared memory object is undefined.

The function **fileno**() examines the argument _stream_ and returns its integer descriptor.

RETURN VALUE

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).

SEE ALSO

open(2), **fclose**(3), **fileno**(3)

NAME

fgetc, fgets, getc, getchar, fputc, fputs, putc, putchar – input and output 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);
int fputc(int _c_**, FILE ***_stream_**);**
int fputs(const char ***_s_, FILE ***_stream_**);**
int putc(int _c_**, FILE ***_stream_**);**
int putchar(int _c_**);**

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_).

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.

fputc() writes the character _c_, cast to an _unsigned char_, to _stream_.

fputs() writes the string _s_ to _stream_, without its terminating null byte ('\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_**).**

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 VALUE

fgetc(), **getc**() and **getchar**() return the character read as an _unsigned char_ cast to an _int_ or **EOF** on end of file or error.

fgets() returns _s_ on success, and NULL on error or when end of file occurs while no characters have been read. **fputc**(), **putc**() and **putchar**() return the character written as an _unsigned char_ cast to an _int_ or **EOF** on error.

fputs() returns a nonnegative number on success, or **EOF** on error.

SEE ALSO

read(2), **write**(2), **ferror**(3), **fgetwc**(3), **fgetws**(3), **fopen**(3), **fread**(3), **fseek**(3), **getline**(3), **getwchar**(3), **scanf**(3), **ungetwc**(3), **write**(2), **ferror**(3), **fopen**(3), **fputwc**(3), **fputws**(3), **fseek**(3), **fwrite**(3), **gets**(3), **putwchar**(3), **scanf**(3), **unlocked_stdio**(3)

NAME

stat, fstat, lstat – get file status

SYNOPSIS

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

int stat(const char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
int lstat(const char *path, struct stat *buf);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

lstat(): _BSD_SOURCE || _XOPEN_SOURCE >= 500

DESCRIPTION

These functions return information about a file. No permissions are required on the file itself, but — in the case of **stat**() and **lstat**() — execute (search) permission is required on all of the directories in *path* that lead to the file.

stat() stats the file pointed to by *path* and fills in *buf*.

lstat() is identical to **stat**(), except that if *path* is a symbolic link, then the link itself is stat-ed, not the file that it refers to.

fstat() is identical to **stat**(), except that the file to be stat-ed is specified by the file descriptor *fd*.

All of these system calls return a *stat* structure, which contains the following fields:

```
struct stat {
      dev_t      st_dev;       /* ID of device containing file  */
      ino_t      st_ino;       /* inode number                  */
      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 ID (if special file)   */
      off_t      st_size;      /* total size, in bytes          */
      blksize_t  st_blksize;   /* blocksize for file system I/O */
      blkcnt_t   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 status change    */
};
```

The *st_dev* field describes the device on which this file resides.

The *st_rdev* field describes the device that this file (inode) represents.

The *st_size* field gives the size of the file (if it is a regular file or a symbolic link) in bytes. The size of a symlink is the length of the pathname it contains, without a trailing null byte.

The *st_blocks* field indicates the number of blocks allocated to the file, 512-byte units. (This may be smaller than *st_size*/512 when the file has holes.)

The *st_blksize* field gives the "preferred" blocksize for efficient file system I/O. (Writing to a file in smaller chunks may cause an inefficient read-modify-rewrite.)

Not all of the Linux file systems implement all of the time fields. Some file system types allow mounting in such a way that file accesses do not cause an update of the *st_atime* field. (See "noatime" in **mount**(8).)

The field *st_atime* is changed by file accesses, for example, by **execve**(2), **mknod**(2), **pipe**(2), **utime**(2) and **read**(2) (of more than zero bytes). Other routines, like **mmap**(2), may or may not update *st_atime*.

The field *st_mtime* is changed by file modifications, for example, by **mknod**(2), **truncate**(2), **utime**(2) and **write**(2) (of more than zero bytes). Moreover, *st_mtime* of a directory is changed by the creation or deletion of files in that directory. The *st_mtime* field is *not* changed for changes in owner, group, hard link count, or mode.

The field *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 using the *st_mode* field:

S_ISREG(m)	is it a regular file?
S_ISDIR(m)	directory?
S_ISCHR(m)	character device?
S_ISBLK(m)	block device?
S_ISFIFO(m)	FIFO (named pipe)?
S_ISLNK(m)	symbolic link? (Not in POSIX.1-1996.)
S_ISSOCK(m)	socket? (Not in POSIX.1-1996.)

RETURN VALUE

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

ERRORS

EACCES
Search permission is denied for one of the directories in the path prefix of *path*. (See also **path_resolution**(7).)

EBADF
fd is bad.

EFAULT
Bad address.

ELOOP
Too many symbolic links encountered while traversing the path.

ENAMETOOLONG
File name too long.

ENOENT
A component of the path *path* does not exist, or the path is an empty string.

ENOMEM
Out of memory (i.e., kernel memory).

ENOTDIR
A component of the path is not a directory.

SEE ALSO

access(2), **chmod**(2), **chown**(2), **fstatat**(2), **readlink**(2), **utime**(2), **capabilities**(7), **symlink**(7)