NAME

chmod, **fchmod**, **lchmod**, **fchmodat** - change mode of file

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <sys/stat.h>

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

int
fchmod(*int fd, mode_t mode*);

int
lchmod(*const char *path, mode_t mode*);

int
fchmodat(*int fd, const char *path, mode_t mode, int flag*);

DESCRIPTION

The file permission bits of the file named specified by *path* or referenced by the file descriptor *fd* are changed to *mode*. The **chmod**() system call verifies that the process owner (user) either owns the file specified by *path* (or *fd*), or is the super-user. The **chmod**() system call follows symbolic links to operate on the target of the link rather than the link itself.

The **lchmod**() system call is similar to **chmod**() but does not follow symbolic links.

The **fchmodat**() is equivalent to either **chmod**() or **lchmod**() depending on the *flag* except in the case where *path* specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. The values for the *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in <*fcntl.h*>:

AT_SYMLINK_NOFOLLOW
: If *path* names a symbolic link, then the mode of the symbolic link is changed.

AT_RESOLVE_BENEATH
: Only walk paths below the directory specified by the *fd* descriptor. See the description of the O_RESOLVE_BENEATH flag in the open(2) manual page.

AT_EMPTY_PATH

If the *path* argument is an empty string, operate on the file or directory referenced by the descriptor *fd*. If *fd* is equal to AT_FDCWD, operate on the current working directory.

If **fchmodat**() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory is used. If also *flag* is zero, the behavior is identical to a call to **chmod**().

A mode is created from *or'd* permission bit masks defined in <*sys/stat.h*>:

```
#define S_IRWXU 0000700    /* RWX mask for owner */
#define S_IRUSR 0000400    /* R for owner */
#define S_IWUSR 0000200    /* W for owner */
#define S_IXUSR 0000100    /* X for owner */

#define S_IRWXG 0000070    /* RWX mask for group */
#define S_IRGRP 0000040    /* R for group */
#define S_IWGRP 0000020    /* W for group */
#define S_IXGRP 0000010    /* X for group */

#define S_IRWXO 0000007    /* RWX mask for other */
#define S_IROTH 0000004    /* R for other */
#define S_IWOTH 0000002    /* W for other */
#define S_IXOTH 0000001    /* X for other */

#define S_ISUID 0004000    /* set user id on execution */
#define S_ISGID 0002000    /* set group id on execution */
#define S_ISVTX 0001000    /* sticky bit */
```

The non-standard S_ISTXT is a synonym for S_ISVTX.

The FreeBSD VM system totally ignores the sticky bit (S_ISVTX) for executables. On UFS-based file systems (FFS, LFS) the sticky bit may only be set upon directories.

If mode S_ISVTX (the 'sticky bit') is set on a directory, an unprivileged user may not delete or rename files of other users in that directory. The sticky bit may be set by any user on a directory which the user owns or has appropriate permissions. For more details of the properties of the sticky bit, see sticky(7).

If mode ISUID (set UID) is set on a directory, and the MNT_SUIDDIR option was used in the mount of the file system, then the owner of any new files and sub-directories created within this directory are set to be the same as the owner of that directory. If this function is enabled, new directories will inherit the

bit from their parents. Execute bits are removed from the file, and it will not be given to root. This behavior does not change the requirements for the user to be allowed to write the file, but only the eventual owner after it has been created. Group inheritance is not affected.

This feature is designed for use on fileservers serving PC users via ftp, SAMBA, or netatalk. It provides security holes for shell users and as such should not be used on shell machines, especially on home directories. This option requires the SUIDDIR option in the kernel to work. Only UFS file systems support this option. For more details of the suiddir mount option, see mount(8).

Writing or changing the owner of a file turns off the set-user-id and set-group-id bits unless the user is the super-user. This makes the system somewhat more secure by protecting set-user-id (set-group-id) files from remaining set-user-id (set-group-id) if they are modified, at the expense of a degree of compatibility.

RETURN VALUES

Upon successful completion, the value 0 is returned; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

The **chmod**() system call will fail and the file mode will be unchanged if:

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

[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.

[ELOOP] Too many symbolic links were encountered in translating the pathname.

[EPERM] The effective user ID does not match the owner of the file and the effective user ID is not the super-user.

[EPERM] The effective user ID is not the super-user, the effective user ID do match the owner of the file, but the group ID of the file does not match the effective group ID nor one of the supplementary group IDs.

[EPERM] The named file has its immutable or append-only flag set, see the chflags(2) manual page for more information.

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

[EFAULT] The *path* argument points outside the process's allocated address space.

[EIO] An I/O error occurred while reading from or writing to the file system.

[EINTEGRITY] Corrupted data was detected while reading from the file system.

[EFTYPE] The effective user ID is not the super-user, the mode includes the sticky bit (S_ISVTX), and path does not refer to a directory.

The **fchmod**() system call will fail if:

[EBADF] The descriptor is not valid.

[EINVAL] The *fd* argument refers to a socket, not to a file.

[EROFS] The file resides on a read-only file system.

[EIO] An I/O error occurred while reading from or writing to the file system.

[EINTEGRITY] Corrupted data was detected while reading from the file system.

In addition to the **chmod**() errors, **fchmodat**() fails if:

[EBADF] The *path* argument does not specify an absolute path and the *fd* argument is neither *AT_FDCWD* nor a valid file descriptor open for searching.

[EINVAL] The value of the *flag* argument is not valid.

[ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a file descriptor associated with a directory.

[ENOTCAPABLE] *path* is an absolute path, or contained a ".." component leading to a directory outside of the directory hierarchy specified by *fd*, and the process is in capability mode or the AT_RESOLVE_BENEATH flag was specified.

SEE ALSO

chmod(1), chflags(2), chown(2), open(2), stat(2), sticky(7)

STANDARDS

The **chmod**() system call is expected to conform to IEEE Std 1003.1-1990 ("POSIX.1"), except for the return of EFTYPE. The S_ISVTX bit on directories is expected to conform to Version 3 of the Single UNIX Specification ("SUSv3"). The **fchmodat**() system call is expected to conform to IEEE Std 1003.1-2008 ("POSIX.1").

HISTORY

The **chmod**() function appeared in Version 1 AT&T UNIX. The **fchmod**() system call appeared in 4.2BSD. The **lchmod**() system call appeared in FreeBSD 3.0. The **fchmodat**() system call appeared in FreeBSD 8.0.