NAME
>   chmod, fchmod, fchmodat - change permissions of a file

SYNOPSIS
>   **#include <sys/stat.h>**
>
>   **int chmod(const char ***pathname**, mode_t** mode**);**
>   **int fchmod(int** fd**, mode_t** mode**);**
>
>   **#include <fcntl.h>** /* Definition of AT_* constants */
>   **#include <sys/stat.h>**
>
>   **int fchmodat(int** dirfd**, const char ***pathname**, mode_t** mode**, int** flags**);**

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

>   **fchmod**():
>   _BSD_SOURCE || _XOPEN_SOURCE >= 500 ||
>   _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
> || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
>
>   **fchmodat**():
>   Since glibc 2.10:
>   _XOPEN_SOURCE >= 700 || _POSIX_C_SOURCE >= 200809L
>   Before glibc 2.10:
>   _ATFILE_SOURCE

DESCRIPTION
>   The **chmod**() and **fchmod**() system calls change the permissions of a file. They differ only in
>   how the file is specified:

>   * **chmod**() changes the permissions of the file specified whose pathname is given in pathname,
>   which is dereferenced if it is a symbolic link.

>   * **fchmod**() changes the permissions of the file referred to by the open file descriptor fd.

>   The new file permissions are specified in mode, which is a bit mask created by ORing together
>   zero or more of the following:

>   **S_ISUID** (04000) set-user-ID (set process effective user ID on execve(2))

>   **S_ISGID** (02000) set-group-ID (set process effective group ID on execve(2); mandatory locking,
>   as described in fcntl(2); take a new file's group from parent directory, as
>   described in chown(2) and mkdir(2))

>   **S_ISVTX** (01000) sticky bit (restricted deletion flag, as described in unlink(2))

>   **S_IRUSR** (00400) read by owner

>   **S_IWUSR** (00200)
>   write by owner

>   **S_IXUSR** (00100) execute/search by owner (search applies for directories, and means that
>   entries within the directory can be accessed)

>   **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 calling process must match the owner of the file, or the process must be privileged (Linux: it must have the **CAP_FOWNER** capability).

If the calling process is not privileged (Linux: does not have the **CAP_FSETID** capability), 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.

As a security measure, depending on the filesystem, the set-user-ID and set-group-ID execution bits may be turned off if a file is written. (On Linux this occurs if the writing process does not have the **CAP_FSETID** capability.) On some filesystems, only the superuser can set the sticky bit, which may have a special meaning. For the sticky bit, and for set-user-ID and set-group-ID bits on directories, see stat(2).

On NFS filesystems, 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.

fchmodat()

The **fchmodat**() system call operates in exactly the same way as **chmod**(), except for the differences described here.

If the pathname given in *pathname* is relative, then it is interpreted relative to the directory referred to by the file descriptor *dirfd* (rather than relative to the current working directory of the calling process, as is done by **chmod**() for a relative pathname).

If *pathname* is relative and *dirfd* is the special value **AT_FDCWD**, then *pathname* is interpreted relative to the current working directory of the calling process (like **chmod**()).

If *pathname* is absolute, then *dirfd* is ignored.

flags can either be 0, or include the following flag:

AT_SYMLINK_NOFOLLOW

> If *pathname* is a symbolic link, do not dereference it: instead operate on the link itself. This flag is not currently implemented.

See openat(2) for an explanation of the need for **fchmodat**().

RETURN VALUE

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

ERRORS

Depending on the filesystem, errors other than those listed below can be returned.

The more general errors for **chmod**() are listed below:

EACCES

> Search permission is denied on a component of the path prefix. (See also path_resolution(7).)

EFAULT

> *pathname* points outside your accessible address space.

EIO   An I/O error occurred.

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

ENAMETOOLONG
> *pathname* 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.

EPERM
> The effective UID does not match the owner of the file, and the process is not privileged (Linux: it does not have the **CAP_FOWNER** capability).

EROFS
> The named file resides on a read-only filesystem.

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

EBADF
> The file descriptor *fd* is not valid.

EIO   See above.

EPERM
> See above.

EROFS
> See above.

The same errors that occur for **chmod**() can also occur for **fchmodat**(). The following additional errors can occur for **fchmodat**():

EBADF
> *dirfd* is not a valid file descriptor.

EINVAL
> Invalid flag specified in *flags*.

ENOTDIR
> *pathname* is relative and *dirfd* is a file descriptor referring to a file other than a directory.

ENOTSUP
> *flags* specified **AT_SYMLINK_NOFOLLOW**, which is not supported.

VERSIONS
> **fchmodat**() was added to Linux in kernel 2.6.16; library support was added to glibc in version 2.4.

CONFORMING TO
> **chmod**(), **fchmod**(): 4.4BSD, SVr4, POSIX.1-2001i, POSIX.1-2008.

> **fchmodat**(): POSIX.1-2008.

NOTES
C library/kernel ABI differences
> The GNU C library **fchmodat**() wrapper function implements the POSIX-specified interface described in this page. This interface differs from the underlying Linux system call, which does *not* have a *flags* argument.

Glibc notes

On older kernels where **fchmodat**() is unavailable, the glibc wrapper function falls back to the use of **chmod**(). When *pathname* is a relative pathname, glibc constructs a pathname based on the symbolic link in */proc/self/fd* that corresponds to the *dirfd* argument.

SEE ALSO

chown(2), execve(2), open(2), stat(2), path_resolution(7), symlink(7)

COLOPHON

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at http://www.kernel.org/doc/man-pages/.