NAME

chown, fchown, lchown, fchownat - change ownership of a file

SYNOPSIS

```
#include <unistd.h>

int chown(const char *pathname, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);
int lchown(const char *pathname, uid_t owner, gid_t group);

#include <fcntl.h> /* Definition of AT_* constants */
#include <unistd.h>

int fchownat(int dirfd, const char *pathname,
             uid_t owner, gid_t group, int flags);
```

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

fchown(), **lchown**():

_BSD_SOURCE || _XOPEN_SOURCE >= 500 ||
_XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
|| /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L

fchownat():

Since glibc 2.10:
_XOPEN_SOURCE >= 700 || _POSIX_C_SOURCE >= 200809L
Before glibc 2.10:
_ATFILE_SOURCE

DESCRIPTION

These system calls change the owner and group of a file. The **chown**(), **fchown**(), and **lchown**() system calls differ only in how the file is specified:

* **chown**() changes the ownership of the file specified by *pathname*, which is dereferenced if it is a symbolic link.
* **fchown**() changes the ownership of the file referred to by the open file descriptor *fd*.
* **lchown**() is like **chown**(), but does not dereference symbolic links.

Only a privileged process (Linux: one with the **CAP_CHOWN** capability) 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. A privileged process (Linux: with **CAP_CHOWN**) 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 an unprivileged 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 behavior depends on the kernel version. In case of a non-group-executable file (i.e., one for which the **S_IXGRP** bit is not set) the **S_ISGID** bit indicates mandatory locking, and is not cleared by a **chown**().

fchownat()

The **fchownat**() system call operates in exactly the same way as **chown**(), 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 **chown**() 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 **chown**()).

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

The *flags* argument is a bit mask created by ORing together 0 or more of the following values;

AT_EMPTY_PATH (since Linux 2.6.39)
If *pathname* is an empty string, operate on the file referred to by *dirfd* (which may have been obtained using the open(2) **O_PATH** flag). In this case, *dirfd* can refer to any type of file, not just a directory. If*dirfd* is **A T_FDCWD**, the call operates on the current working directory. This flag is Linux-specific; define **_GNU_SOURCE** to obtain its definition.

AT_SYMLINK_NOFOLLOW
If *pathname* is a symbolic link, do not dereference it: instead operate on the link itself, like **lchown**(). (By default, **fchownat**() dereferences symbolic links, like **chown**().)

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

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 **chown**() 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.

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 calling process did not have the required permissions (see above) to change owner and/or group.

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

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

EBADF
The descriptor is not valid.

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

ENOENT
See above.

EPERM
See above.

EROFS
See above.

The same errors that occur for **chown**() can also occur for **fchownat**(). The following additional errors can

occur for **fchownat**():

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.

VERSIONS

fchownat() was added to Linux in kernel 2.6.16; library support was added to glibc in version 2.4.

CONFORMING TO

chown(), **fchown**(), **lchown**(): 4.4BSD, SVr4, POSIX.1-2001, POSIX.1-2008.

The 4.4BSD version can be used only by the superuser (that is, ordinary users cannot give away files).

fchownat(): POSIX.1-2008.

NOTES

Ownership of new files

When a new file is created (by, for example, open(2) or mkdir(2)), its owner is made the same as the filesystem user ID of the creating process. The group of the file depends on a range of factors, including the type of filesystem, the options used to mount the filesystem, and whether or not the set-group-ID permission bit is enabled on the parent directory. If the filesystem supports the-*o grpid* (or , synonymously *-o bsdgroups*) and *-o nogrpid* (or, synonymously *-o sysvgroups*) mount(8) options, then the rules are as follows:

* If the filesystem is mounted with *-o grpid*, then the group of a new file is made the same as that of the parent directory.
* If the filesystem is mounted with *-o nogrpid* and the set-group-ID bit is disabled on the parent directory, then the group of a new file is made the same as the process's filesystem GID.
* If the filesystem is mounted with *-o nogrpid* and the set-group-ID bit is enabled on the parent directory, then the group of a new file is made the same as that of the parent directory.

As at Linux 2.6.25, the *-o grpid* and *-o nogrpid* mount options are supported by ext2, ext3, ext4, and XFS. Filesystems that don't support these mount options follow the *-o nogrpid* rules.

Glibc notes

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

NFS

The **chown**() semantics are deliberately violated on NFS filesystems 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.

Historical details

The original Linux **chown**(), **fchown**(), and **lchown**() system calls supported only 16-bit user and group IDs. Subsequently, Linux 2.4 added **chown32**(), **fchown32**(), and **lchown32**(), supporting 32-bit IDs. The glibc **chown**(), **fchown**(), and **lchown**() wrapper functions transparently deal with the variations across kernel versions.

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.

EXAMPLE

The following program changes the ownership of the file named in its second command-line argument to the value specified in its first command-line argument. The new owner can be specified either as a numeric user ID, or as a username (which is converted to a user ID by using getpwnam(3) to perform a lookup in the system password file).

Program source

```
#include <pwd.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
uid_t uid;
struct passwd *pwd;
char *endptr;

if (argc != 3 || argv[1][0] == '\0') {
fprintf(stderr, "%s <owner> <file>\n", argv[0]);
exit(EXIT_FAILURE);
}

uid = strtol(argv[1], &endptr, 10); /* Allow a numeric string */

if (*endptr != '\0') { /* Was not pure numeric string */
pwd = getpwnam(argv[1]); /* Try getting UID for username */
if (pwd == NULL) {
perror("getpwnam");
exit(EXIT_FAILURE);
}

uid = pwd->pw_uid;
}

if (chown(argv[2], uid, -1) == -1) {
perror("chown");
exit(EXIT_FAILURE);
}

exit(EXIT_SUCCESS);
}
```

SEE ALSO

chmod(2), flock(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/.