NAME

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

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

```
#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);

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

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

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

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

```
/* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
    || _XOPEN_SOURCE >= 500
    || /* glibc <= 2.19: */ _BSD_SOURCE
```

fchownat():

```
Since glibc 2.10:
    _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 *path*, 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 is 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, and since Linux 2.2.13, root is treated like other users. 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**().

When the owner or group of an executable file is changed (by any user), all capability sets for the file are cleared.

fchownat()

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

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

If *path* 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 *path* 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 *path* 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 to indicate the error.

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).)

EBADF
: (**fchown**()) *fd* is not a valid open file descriptor.

EBADF
: (**fchownat**()) *path* is relative but *dirfd* is neither **AT_FDCWD** nor a valid file descriptor.

EFAULT
: *path* points outside your accessible address space.

EINVAL
: (**fchownat**()) Invalid flag specified in *flags*.

EIO (**fchown**()) A low-level I/O error occurred while modifying the inode.

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

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.

ENOTDIR
: (**fchownat**()) *path* is relative and *dirfd* is a file descriptor referring to a file other than a directory.

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

EPERM
The file is marked immutable or append-only. (See**FS_IOC_SETFLA GS**(2const).)

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

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

STANDARDS
POSIX.1-2024.

HISTORY
chown()
SVr4, POSIX.1-1988, 4.4BSD.

fchown()
SVr4, 4.4BSD, SUSv1, POSIX.1-2001.

lchown()
SVr4, 4.4BSD, SUSv1, POSIX.1-2001 XSI, POSIX.1-2008.

fchownat()
POSIX.1-2008. Linux 2.6.16, glibc 2.4.

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 mode bit is enabled on the parent directory. If the filesystem supports the**−o gr pid** (or, synonymously **−o bsd-groups**) 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 4.12, 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*path* is relati ve, 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.

Before Linux 2.1.81 (except 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.

EXAMPLES

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

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

    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

chgrp(1), **chown**(1), **chmod**(2), **flock**(2), **path_resolution**(7), **symlink**(7)