NAME

flock − apply or remove an advisory lock on an open file

SYNOPSIS

#include <sys/file.h>

int flock(int *fd***, int** *operation***);**

DESCRIPTION

Apply or remove an advisory lock on the open file specified by *fd*. The argument *operation* is one of the following:

LOCK_SH
Place a shared lock. More than one process may hold a shared lock for a given file at a given time.

LOCK_EX
Place an exclusive lock. Only one process may hold an exclusive lock for a given file at a given time.

LOCK_UN
Remove an existing lock held by this process.

A call to **flock**() may block if an incompatible lock is held by another process. To make a non-blocking request, include **LOCK_NB** (by *OR*ing) with any of the above operations.

A single file may not simultaneously have both shared and exclusive locks.

Locks created by **flock**() are associated with an open file table entry. This means that duplicate file descriptors (created by, for example, **fork**(2) or **dup**(2)) refer to the same lock, and this lock may be modified or released using any of these descriptors. Furthermore, the lock is released either by an explicit **LOCK_UN** operation on any of these duplicate descriptors, or when all such descriptors have been closed.

If a process uses **open**(2) (or similar) to obtain more than one descriptor for the same file, these descriptors are treated independently by **flock**(). An attempt to lock the file using one of these file descriptors may be denied by a lock that the calling process has already placed via another descriptor.

A process may only hold one type of lock (shared or exclusive) on a file. Subsequent **flock**() calls on an already locked file will convert an existing lock to the new lock mode.

Locks created by **flock**() are preserved across an **execve**(2).

A shared or exclusive lock can be placed on a file regardless of the mode in which the file was opened.

RETURN VALUE

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

ERRORS

EBADF
fd is not an open file descriptor.

EINTR
While waiting to acquire a lock, the call was interrupted by delivery of a signal caught by a handler; see **signal**(7).

EINVAL
operation is invalid.

ENOLCK
The kernel ran out of memory for allocating lock records.

EWOULDBLOCK
The file is locked and the **LOCK_NB** flag was selected.

CONFORMING TO

4.4BSD (the **flock**() call first appeared in 4.2BSD). A version of **flock**(), possibly implemented in terms of **fcntl**(2), appears on most Unix systems.

NOTES

Since kernel 2.0, **flock**() is implemented as a system call in its own right rather than being emulated in the GNU C library as a call to **fcntl**(2). This yields classical BSD semantics: there is no interaction between the types of lock placed by **flock**() and **fcntl**(2), and **flock**() does not detect deadlock. (Note, however, that on some modern BSDs, **flock**() and **fcntl**(2) locks *do* interact with one another.)

In Linux kernels up to 2.6.11, **flock**() does not lock files over NFS (i.e., the scope of locks was limited to the local system). Instead, one could use **fcntl**(2) byte-range locking, which does work over NFS, given a sufficiently recent version of Linux and a server which supports locking. Since Linux 2.6.12, NFS clients support **flock**() locks by emulating them as byte-range locks on the entire file. This means that **fcntl**(2) and **flock**() locks *do* interact with one another over NFS. Since Linux 2.6.37, the kernel supports a compatibility mode that allows **flock**() locks (and also **fcntl**(2) byte region locks) to be treated as local; see the discussion of the *local_lock* option in **nfs**(5).

flock() places advisory locks only; given suitable permissions on a file, a process is free to ignore the use of **flock**() and perform I/O on the file.

flock() and **fcntl**(2) locks have different semantics with respect to forked processes and **dup**(2). On systems that implement **flock**() using **fcntl**(2), the semantics of **flock**() will be different from those described in this manual page.

Converting a lock (shared to exclusive, or vice versa) is not guaranteed to be atomic: the existing lock is first removed, and then a new lock is established. Between these two steps, a pending lock request by another process may be granted, with the result that the conversion either blocks, or fails if **LOCK_NB** was specified. (This is the original BSD behavior, and occurs on many other implementations.)

SEE ALSO

close(2), **dup**(2), **execve**(2), **fcntl**(2), **fork**(2), **open**(2), **lockf**(3)

See also *Documentation/filesystem/locks.txt* in the kernel source (*Documentation/locks.txt* in older kernels).

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.