



Linux Ubuntu 26.04 Manual Pages on command 'flock.2'

\$ man flock.2

flock(2) System Calls Manual flock(2)

NAME

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

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <sys/file.h>
```

```
int flock(int fd, int op);
```

DESCRIPTION

Apply or remove an advisory lock on the open file specified by fd. The argument op 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 nonblocking request, include LOCK_NB (by ORing) 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 description (see open(2)). 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 file descrip?

tors. Furthermore, the lock is released either by an explicit LOCK_UN operation on any of

these duplicate file descriptors, or when all such file descriptors have been closed.

If a process uses `open(2)` (or similar) to obtain more than one file descriptor for the same file, these file 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 file descriptor.

A process may hold only 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 to indicate the error.

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 `op` is invalid.

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

EWOLDBLOCK

The file is locked and the `LOCK_NB` flag was selected.

VERSIONS

Since Linux 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)`. With this implementation, 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 systems, such as the modern BSDs, `flock()` and `fcntl(2)` locks do interact with one another.)

CIFS details

Up to Linux 5.4, `flock()` is not propagated over SMB. A file with such locks will not appear locked for remote clients.

Since Linux 5.5, `flock()` locks are emulated with SMB byte-range locks on the entire file.

Similarly to NFS, this means that `fcntl(2)` and `flock()` locks interact with one another. An?

other important side-effect is that the locks are not advisory anymore: any IO on a locked

file will always fail with EACCES when done from a separate file descriptor. This difference originates from the design of locks in the SMB protocol, which provides mandatory locking semantics.

Remote and mandatory locking semantics may vary with SMB protocol, mount options and server type. See mount.cifs(8) for additional information.

STANDARDS

BSD.

HISTORY

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.

NFS details

Up to Linux 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 fcntl(2) byte-range locks on the entire file. This means that fcntl(2) and flock() locks do interact with one another over NFS. It also means that in order to place an exclusive lock, the file must be opened for writing.

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).

NOTES

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.)

Release of a lock when a file descriptor is closed is not sequenced after all observable effects of `close(2)`. For example, if one process writes a file while holding an exclusive lock, then closes that file, and another process blocks placing a shared lock on that file to wait until it is closed, it may observe that a subsequent `execve(2)` of that file fails with `ETXTBSY`, and `umount(2)` of its underlying filesystem fails with `EBUSY`, as if the file were still open in the first process.

SEE ALSO

`flock(1)`, `close(2)`, `dup(2)`, `execve(2)`, `fcntl(2)`, `fork(2)`, `open(2)`, `lockf(3)`, `locks(8)`

Documentation/filesystems/locks.rst in the Linux kernel source tree

Linux man-pages 6.17

2026-02-08

`flock(2)`