NAME

lockf - record locking on files

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <unistd.h>

int
lockf(*int fd, int function, off_t size*);

DESCRIPTION

The **lockf**() function allows sections of a file to be locked with advisory-mode locks. Calls to **lockf**() from other processes which attempt to lock the locked file section will either return an error value or block until the section becomes unlocked. All the locks for a process are removed when the process terminates.

The argument *fd* is an open file descriptor. The file descriptor must have been opened either for write-only (O_WRONLY) or read/write (O_RDWR) operation.

The *function* argument is a control value which specifies the action to be taken. The permissible values for *function* are as follows:

Function	**Description**
F_ULOCK	unlock locked sections
F_LOCK	lock a section for exclusive use
F_TLOCK	test and lock a section for exclusive use
F_TEST	test a section for locks by other processes

F_ULOCK removes locks from a section of the file; F_LOCK and F_TLOCK both lock a section of a file if the section is available; F_TEST detects if a lock by another process is present on the specified section.

The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be locked or unlocked starts at the current offset in the file and extends forward for a positive size or backward for a negative size (the preceding bytes up to but not including the current offset). However, it is not permitted to lock a section that starts or extends before the beginning of the file. If *size* is 0, the section from the current offset through the largest possible file offset is locked (that is, from the current offset through the present or any future end-of-file).

The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent locked sections would occur, the sections are combined into a single locked section. If the request would cause the number of locks to exceed a system-imposed limit, the request will fail.

F_LOCK and F_TLOCK requests differ only by the action taken if the section is not available. F_LOCK blocks the calling process until the section is available. F_TLOCK makes the function fail if the section is already locked by another process.

File locks are released on first close by the locking process of any file descriptor for the file.

F_ULOCK requests release (wholly or in part) one or more locked sections controlled by the process. Locked sections will be unlocked starting at the current file offset through *size* bytes or to the end of file if size is 0. When all of a locked section is not released (that is, when the beginning or end of the area to be unlocked falls within a locked section), the remaining portions of that section are still locked by the process. Releasing the center portion of a locked section will cause the remaining locked beginning and end portions to become two separate locked sections. If the request would cause the number of locks in the system to exceed a system-imposed limit, the request will fail.

An F_ULOCK request in which size is non-zero and the offset of the last byte of the requested section is the maximum value for an object of type off_t, when the process has an existing lock in which size is 0 and which includes the last byte of the requested section, will be treated as a request to unlock from the start of the requested section with a size equal to 0. Otherwise an F_ULOCK request will attempt to unlock only the requested section.

A potential for deadlock occurs if a process controlling a locked region is put to sleep by attempting to lock the locked region of another process. This implementation detects that sleeping until a locked region is unlocked would cause a deadlock and fails with an EDEADLK error.

The **lockf**(), fcntl(2), and flock(2) locks are compatible. Processes using different locking interfaces can cooperate over the same file safely. However, only one of such interfaces should be used within the same process. If a file is locked by a process through flock(2), any record within the file will be seen as locked from the viewpoint of another process using fcntl(2) or **lockf**(), and vice versa.

Blocking on a section is interrupted by any signal.

RETURN VALUES

The **lockf**() function returns the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error. In the case of a failure, existing locks are not changed.

ERRORS

The **lockf**() function will fail if:

[EAGAIN] The argument *function* is F_TLOCK or F_TEST and the section is already locked by another process.

[EBADF] The argument *fd* is not a valid open file descriptor.

The argument *function* is F_LOCK or F_TLOCK, and *fd* is not a valid file descriptor open for writing.

[EDEADLK] The argument *function* is F_LOCK and a deadlock is detected.

[EINTR] The argument *function* is F_LOCK and **lockf**() was interrupted by the delivery of a signal.

[EINVAL] The argument *function* is not one of F_ULOCK, F_LOCK, F_TLOCK or F_TEST.

The argument *fd* refers to a file that does not support locking.

[ENOLCK] The argument *function* is F_ULOCK, F_LOCK or F_TLOCK, and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.

SEE ALSO

fcntl(2), flock(2)

STANDARDS

The **lockf**() function conforms to X/Open Portability Guide Issue 4, Version 2 ("XPG4.2").