Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'lockf.3p' command

$ man lockf.3p

LOCKF(3P) POSIX Programmer's Manual LOCKF(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lockf ? record locking on files

SYNOPSIS

```
#include <unistd.h>
int lockf(int fildes, int function, off_t size);
```

DESCRIPTION

The lockf() function shall lock sections of a file with advisory-mode locks. Calls to lockf() from threads in other processes which attempt to lock the locked file section shall either return an error value or block until the section becomes unlocked. All the locks for a process are removed when the process terminates. Record locking with lockf() shall be supported for regular files and may be supported for other files.

The fildes argument is an open file descriptor. To establish a lock with this function, the file descriptor shall be opened with write-only permission (O_WRONLY) or with read/write permission (O_RDWR).

The function argument is a control value which specifies the action to

be taken. The permissible values for function are defined in <unistd.h> 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_TEST shall detect if a lock by another process is present on the specified section.

F_LOCK and F_TLOCK shall both lock a section of a file if the section is available.

F_ULOCK shall remove locks from a section of the file.

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). If size is 0, the section from the current offset through the largest possible file offset shall be locked (that is, from the current offset through the present or any future end-of-file). An area need not be allocated to the file to be locked because locks may exist past the 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 shall be combined into a single locked section. If the request would cause the number of locks to exceed a system-imposed limit, the request shall fail.

F_LOCK and F_TLOCK requests differ only by the action taken if the section is not available. F_LOCK shall block the calling thread until the section is available. F_TLOCK shall cause the function to fail if the

section is already locked by another process.

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

F_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the process. Locked sections shall be unlocked starting at the current file offset through size bytes or to the end-of-file if size is (off_t)0. When all of a locked section is not re? leased (that is, when the beginning or end of the area to be unlocked falls within a locked section), the remaining portions of that section shall remain locked by the process. Releasing the center portion of a locked section shall cause the remaining locked beginning and end por? tions 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 shall fail.

A potential for deadlock occurs if the threads of a process controlling a locked section are blocked by accessing a locked section of another process. If the system detects that deadlock would occur, lockf() shall fail with an [EDEADLK] error.

The interaction between fcntl() and lockf() locks is unspecified.

Blocking on a section shall be interrupted by any signal.

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, shall 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 shall attempt to unlock only the requested section.

Attempting to lock a section of a file that is associated with a buffered stream produces unspecified results.

RETURN VALUE

Upon successful completion, lockf() shall return 0. Otherwise, it shall return -1, set errno to indicate an error, and existing locks shall not be changed.

ERRORS

The lockf() function shall fail if:

EBADF The fildes argument is not a valid open file descriptor; or function is F_LOCK or F_TLOCK and fildes is not a valid file de? scriptor open for writing.

EACCES or EAGAIN

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

EDEADLK

The function argument is F_LOCK and a deadlock is detected.

EINTR A signal was caught during execution of the function.

EINVAL The function argument is not one of F_LOCK, F_TLOCK, F_TEST, or F_ULOCK; or size plus the current file offset is less than 0.

EOVERFLOW

The offset of the first, or if size is not 0 then the last, byte in the requested section cannot be represented correctly in an object of type off_t.

The lockf() function may fail if:

EAGAIN The function argument is F_LOCK or F_TLOCK and the file is mapped with mmap().

EDEADLK or ENOLCK

The function argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request would cause the number of locks to exceed a system-im? posed limit.

EOPNOTSUPP or EINVAL

The implementation does not support the locking of files of the type indicated by the fildes argument.

The following sections are informative.

EXAMPLES

Locking a Portion of a File

In the following example, a file named /home/cnd/mod1 is being modi? fied. Other processes that use locking are prevented from changing it during this process. Only the first 10000 bytes are locked, and the

lock call fails if another process has any part of this area locked al?

ready.

```
#include <fcntl.h>
#include <unistd.h>
int fildes;
int status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status = lockf(fildes, F_TLOCK, (off_t)10000);
```

APPLICATION USAGE

Record-locking should not be used in combination with the fopen(), fread(), fwrite(), and other stdio functions. Instead, the more primi? tive, non-buffered functions (such as open()) should be used. Unex? pected results may occur in processes that do buffering in the user ad? dress space. The process may later read/write data which is/was locked. The stdio functions are the most common source of unexpected buffering.

The alarm() function may be used to provide a timeout facility in ap? plications requiring it.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), chmod(), close(), creat(), fcntl(), fopen(), mmap(), open(), read(), write()

The Base Definitions volume of POSIX.1?2017, <unistd.h>

COPYRIGHT

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .