

lockf() — Record Locking on Files

Standards

Standards / Extensions	C or C++	Dependencies
XPG4.2 Single UNIX Specification, Version 3	both	

Format

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <unistd.h>

int lockf(int filedes, int function, off_t size);
```

General 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. Record locking with `lockf()` is supported for regular files.

The *filedes* argument is an open file descriptor. The file descriptor must have been opened with a write-only permission (`O_WRONLY`) or with read/write permission (`O_RDWR`) to establish a lock with this function.

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
<code>F_ULOCK</code>	unlock locked sections
<code>F_LOCK</code>	lock a section for exclusive use
<code>F_TLOCK</code>	test and lock a section for exclusive use
<code>F_TEST</code>	test a section for locks by other processes

`F_TEST` detects if a lock by another process is present on the specified section; `F_LOCK` and `F_TLOCK` both lock a section of a file if the section is available; `F_ULOCK` removes 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 is locked (that is, from the current offset through the present or any future End Of File (EOF)). 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 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 may 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 (EOF) if *size* is (off_t)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.

A potential for deadlock occurs if a process controlling a locked section is blocked by accessing another process's locked section. If the system detects that a deadlock would occur, lockf() will fail with an EDEADLK error.

Locks obtained by lockf() are controlled by the same facility controlling locks obtained by fcntl().

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

Blocking on a section is interrupted by any signal.

Large Files for HFS

Note:

Large Files for HFS behavior is automatic for AMODE 64 applications

Applications that are compiled with the option LONGLVL(LONGLONG) and also define the Feature Test Macro (FTM) _LARGE_FILES before any headers are included will enable this function to operate on HFS files that are larger than 2 gig-1 in size. File size and offset fields will be enlarged to 63 bits in width so any other function operating on this file will have to be enabled with the same FTM.

Returned Value

If successful, lockf() returns 0.

If unsuccessful, existing locks are not changed, lockf() returns -1, and sets errno to one of the following values:

Error Code

	Description
EACCES or EAGAIN	The <i>function</i> argument is F_TLOCK or F_TEST and the section is already locked by another process
EBADF	The <i>filedes</i> argument is not a valid open file descriptor; or <i>function</i> is F_LOCK or F_TLOCK and <i>filedes</i> is not a valid file descriptor open for writing.
EDEADLK	The <i>function</i> argument is F_LOCK and a deadlock is detected.
EINTR	A signal was caught during execution of the function.
EOVERFLOW	The offset of the first, or if <i>size</i> is not 0 then the last, byte in the requested section cannot be represented correctly in an object of type off_t.