NAME

lockf – record locking on files

SYNOPSIS

#include <unistd.h>

int lockf(int *fildes*, **int** *function*, **long** *size***);**

MT-LEVEL

MT-Safe

DESCRIPTION

lockf( ) allows sections of a file to be locked; advisory or mandatory write locks depending on the mode bits of the file (see **chmod**(2)). Locking calls from other processes that attempt to lock the locked file section will either return an error value or be put to sleep until the resource becomes unlocked. All the locks for a process are removed when the process terminates. (See **fcntl**(2) for more information about record locking.)

fildes is an open file descriptor. The file descriptor must have **O_WRONLY** or **O_RDWR** permission in order to establish locks with this function call.

function is a control value that specifies the action to be taken. The permissible values for *function* are defined in **<unistd.h>** as follows:

```
#define    F_ULOCK    0    /* unlock previously locked section */
#define    F_LOCK     1    /* lock section for exclusive use */
#define    F_TLOCK    2    /* test & lock section for exclusive use */
#define    F_TEST     3    /* test section for other locks */
```

All other values of *function* are reserved for future extensions and will result in an error return if not implemented.

F_TEST is used to detect 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.

size is the number of contiguous bytes to be locked or unlocked. The resource to be locked or unlocked starts at the current offset in the file and extends forward for a positive size and backward for a negative size (the preceding bytes up to but not including the current offset). If *size* is zero, the section from the current offset through the largest file offset is 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 in order to be locked as such 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. Locked sections will be unlocked starting at the the point of the offset through *size* bytes or to the end of file if *size* is (**off_t**) 0. When this situation occurs, or if this situation occurs in adjacent sections, the sections are combined into a single section. If the request requires that a new element be added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.

F_LOCK and **F_TLOCK** requests differ only by the action taken if the resource is not available. **F_LOCK** will cause the calling process to sleep until the resource is available. **F_TLOCK** will cause the function to return a −1 and set **errno** to **EAGAIN** if the section is already locked by another process.

F_ULOCK requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an **errno** is set to **EDEADLK** and the requested section is not released.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by requesting another process's locked resource. Thus calls to **lockf( )** or **fcntl**(2) scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The **alarm**(2) function may be used to provide a timeout facility in applications that require this facility.

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

ERRORS

lockf( ) will fail if one or more of the following are true:

EAGAIN *cmd* is **F_TLOCK** or **F_TEST** and the section is already locked by another process.

EBADF *fildes* is not a valid open descriptor.

ECOMM *fildes* is on a remote machine and the link to that machine is no longer active.

EDEADLK *cmd* is **F_LOCK** and a deadlock occurred.

EDEADLK *cmd* is **F_LOCK**, **F_TLOCK**, or **F_ULOCK** and the number of entries in the lock table exceeded the number allocated on the system.

SEE ALSO

intro(2), **alarm**(2), **chmod**(2), **close**(2), **creat**(2), **fcntl**(2), **open**(2), **read**(2), **write**(2)

NOTES

Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data that is/was locked. The standard I/O package is the most common source of unexpected buffering.

In the past, the variable **errno** was set to **EACCES** rather than **EAGAIN.** When a section of a file is already locked by another process, portable application programs should expect and test for either value.