NAME

lockg - apply, test or remove a POSIX group lock on an open file

SYNOPSIS

```
#include <unistd.h>

int lockg(int fd, int cmd, lgid_t *lgid);
```

DESCRIPTION

Apply, test, remove, or join a POSIX group lock on an open file. Group locks are exclusive, whole-file locks that limit file access to a specified group of processes. The file is specified by fd, a file descriptor open for writing and the action by cmd.

The first process to call lockg() passes a cmd of F_LOCK and an initialized value for lgid. Obtaining the lock is performed exactly as though a lockf() with pos of 0 and len of 0 were used (i.e. defining a lock section that encompasses a region from byte position zero to present and future end-of-tile positions). An opaque lock group id is returned in lgid. This lgid may be passed through external means (e.g. message passing or shared memory communication) to other processes for the purpose of allowing them to join the group lock.

Processes wishing to join the group lock call lockg() with a cmd of F_LOCK and the lgid returned to the first process. On success this process has registered itself as a member of the group of the group lock.

Valid operations are given below:

F_LOCK Set an exclusive lock on the specified section of the file. If (part of) this section is already locked, the call blocks until the previous lock is released. If this section overlaps an earlier locked section, both are merged. File locks are released as soon as the process holding the locks closes some file descriptor for the file. A child process does not inherit these locks.

F_TLOCK
Same as F_LOCK but the call never blocks and returns an error instead if the file is already locked.

F_ULOCK
Unlock the indicated file. If any process in the group performs a F_ULOCK operation, then the file is unlocked for all group members (i.e. the lock is destroyed). It is not possible for the locked session to be split, because group locks apply to the entire file.

F_TEST Test the lock: return 0 if the specified section is unlocked or locked by this process; return -1, set errno to EACCES, if another process holds a lock.

If any process unlocks a file, calls close() on the fd, or terminates, then the group lock is destroyed. Subsequent I/O operations by any lck group member on their corresponding fd will return ENOLCK to indicate to the process that the lock is no longer valid.

RETURN VALUE
On success, zero is returned.

On error, -1 is returned, and errno is set appropriately.

ERRORS
EAGAIN The file is locked and F_TLOCK or F_TEST was specified, or the operation is prohibited because the file has been memory-mapped by another process.

EBADF fd is not an open file descriptor.

EDEADLK
The command was T_LOCK and this lock operation would cause a deadlock.

EINVAL An invalid operation was specified in fd.

ENOLCK Too many segment locks open, lock table is full.

CONFORMING TO

The whims of Lee and Rob, with Gary watching out of the corner of his eye.

NOTES

lockg() honors lockf() locks as well.

Child processes do not inherit lockg() locks. A child process would need to call lockg() with the appropriate lgid in order to join the group lock, prior to any file access.

The lgid parameter may be set to LOCKG_INITIALIZER for static initialization, or the macro LOCKG_INIT(lgid) may be used at runtime for initialization purposes.

There is currently no mechanism through which a process may remove itself from the lock group without destroying the group lock.

Because of the semantics of group locks with respect to processes terminating and calling close(), users are encouraged to externally synchronize (e.g. using message passing or shared memory communication) prior to any F_ULOCK operation. This will ensure that all processes have completed I/O before the group lock is destroyed.

SEE ALSO

lockf(3), fcntl(2), flock(2)