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 'flockfile.3p' command

$ man flockfile.3p

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

flockfile, ftrylockfile, funlockfile ? stdio locking functions

SYNOPSIS

```
#include <stdio.h>
void flockfile(FILE *file);
int ftrylockfile(FILE *file);
void funlockfile(FILE *file);
```

DESCRIPTION

These functions shall provide for explicit application-level locking of stdio (FILE *) objects. These functions can be used by a thread to de? lineate a sequence of I/O statements that are executed as a unit.

The flockfile() function shall acquire for a thread ownership of a (FILE *) object.

The ftrylockfile() function shall acquire for a thread ownership of a (FILE *) object if the object is available; ftrylockfile() is a non-blocking version of flockfile().

The funlockfile() function shall relinquish the ownership granted to

the thread. The behavior is undefined if a thread other than the cur? rent owner calls the funlockfile() function.

The functions shall behave as if there is a lock count associated with each (FILE *) object. This count is implicitly initialized to zero when the (FILE *) object is created. The (FILE *) object is unlocked when the count is zero. When the count is positive, a single thread owns the (FILE *) object. When the flockfile() function is called, if the count is zero or if the count is positive and the caller owns the (FILE *) object, the count shall be incremented. Otherwise, the calling thread shall be suspended, waiting for the count to return to zero. Each call to funlockfile() shall decrement the count. This allows matching calls to flockfile() (or successful calls to ftrylockfile()) and funlock? file() to be nested.

All functions that reference (FILE *) objects, except those with names ending in _unlocked, shall behave as if they use flockfile() and fun? lockfile() internally to obtain ownership of these (FILE *) objects.

RETURN VALUE

None for flockfile() and funlockfile().

The ftrylockfile() function shall return zero for success and non-zero to indicate that the lock cannot be acquired.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inver? sion, as discussed in the Base Definitions volume of POSIX.1?2017, Sec? tion 3.291, Priority Inversion.

A call to exit() can block until locked streams are unlocked because a thread having ownership of a (FILE*) object blocks all function calls that reference that (FILE*) object (except those with names ending in _unlocked) from other threads, including calls to exit().

RATIONALE

The flockfile() and funlockfile() functions provide an orthogonal mu? tual-exclusion lock for each FILE. The ftrylockfile() function pro? vides a non-blocking attempt to acquire a file lock, analogous to pthread_mutex_trylock().

These locks behave as if they are the same as those used internally by stdio for thread-safety. This both provides thread-safety of these functions without requiring a second level of internal locking and al? lows functions in stdio to be implemented in terms of other stdio func? tions.

Application developers and implementors should be aware that there are potential deadlock problems on FILE objects. For example, the line-buffered flushing semantics of stdio (requested via {_IOLBF}) require that certain input operations sometimes cause the buffered contents of implementation-defined line-buffered output streams to be flushed. If two threads each hold the lock on the other's FILE, deadlock ensues. This type of deadlock can be avoided by acquiring FILE locks in a con? sistent order. In particular, the line-buffered output stream deadlock can typically be avoided by acquiring locks on input streams before locks on output streams if a thread would be acquiring both.

In summary, threads sharing stdio streams with other threads can use flockfile() and funlockfile() to cause sequences of I/O performed by a single thread to be kept bundled. The only case where the use of flock? file() and funlockfile() is required is to provide a scope protecting uses of the *_unlocked functions/macros. This moves the cost/perfor? mance tradeoff to the optimal point.

FUTURE DIRECTIONS

None.

SEE ALSO

exit(), getc_unlocked()

The Base Definitions volume of POSIX.1?2017, Section 3.291, Priority Inversion, <stdio.h>

COPYRIGHT