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 'pthread_mutex_lock.3p' command

$ man pthread_mutex_lock.3p

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

pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock ? lock and unlock a mutex

SYNOPSIS

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

DESCRIPTION

The mutex object referenced by mutex shall be locked by a call to pthread_mutex_lock() that returns zero or [EOWNERDEAD]. If the mutex is already locked by another thread, the calling thread shall block un? til the mutex becomes available. This operation shall return with the mutex object referenced by mutex in the locked state with the calling thread as its owner. If a thread attempts to relock a mutex that it has already locked, pthread_mutex_lock() shall behave as described in the Relock column of the following table. If a thread attempts to unlock a

mutex that it has not locked or a mutex which is unlocked, pthread_mu?
tex_unlock() shall behave as described in the Unlock When Not Owner
column of the following table.

Mutex Type	Robustness	Relock	Unlock When Not Owner
NORMAL	non-robust	deadlock	undefined behavior
NORMAL	robust	deadlock	error returned
ERRORCHECK	either	error returned	error returned
RECURSIVE	either	recursive	error returned
?	?	? (see below)	?
DEFAULT	non-robust	undefined	undefined behavior?
?	?	? behavior?	?
DEFAULT	robust	undefined	error returned
?	?	? behavior?	?

? If the mutex type is PTHREAD_MUTEX_DEFAULT, the behavior of pthread_mutex_lock() may correspond to one of the three other standard mutex types as described in the table above. If it does not correspond to one of those three, the behavior is undefined for the cases marked ?.

Where the table indicates recursive behavior, the mutex shall maintain the concept of a lock count. When a thread successfully acquires a mu? tex for the first time, the lock count shall be set to one. Every time a thread relocks this mutex, the lock count shall be incremented by one. Each time the thread unlocks the mutex, the lock count shall be decremented by one. When the lock count reaches zero, the mutex shall become available for other threads to acquire.

The pthread_mutex_trylock() function shall be equivalent to pthread_mu?
tex_lock(), except that if the mutex object referenced by mutex is cur?
rently locked (by any thread, including the current thread), the call
shall return immediately. If the mutex type is PTHREAD_MUTEX_RECURSIVE
and the mutex is currently owned by the calling thread, the mutex lock
count shall be incremented by one and the pthread_mutex_trylock() func?
tion shall immediately return success.
The pthread_mutex_unlock() function shall release the mutex object ref?
erenced by mutex. The manner in which a mutex is released is dependent
upon the mutex's type attribute. If there are threads blocked on the
mutex object referenced by mutex when pthread_mutex_unlock() is called,
resulting in the mutex becoming available, the scheduling policy shall
determine which thread shall acquire the mutex.
(In the case of PTHREAD_MUTEX_RECURSIVE mutexes, the mutex shall become
available when the count reaches zero and the calling thread no longer
has any locks on this mutex.)
If a signal is delivered to a thread waiting for a mutex, upon return
from the signal handler the thread shall resume waiting for the mutex
as if it was not interrupted.
If mutex is a robust mutex and the process containing the owning thread
terminated while holding the mutex lock, a call to pthread_mutex_lock()
shall return the error value [EOWNERDEAD]. If mutex is a robust mutex
and the owning thread terminated while holding the mutex lock, a call
to pthread_mutex_lock() may return the error value [EOWNERDEAD] even if
the process in which the owning thread resides has not terminated. In
these cases, the mutex is locked by the thread but the state it pro?
tects is marked as inconsistent. The application should ensure that the
state is made consistent for reuse and when that is complete call
pthread_mutex_consistent(). If the application is unable to recover
the state, it should unlock the mutex without a prior call to
pthread_mutex_consistent(), after which the mutex is marked permanently
unusable.
If mutex does not refer to an initialized mutex object, the behavior of

pthread_mutex_lock(), pthread_mutex_trylock(), and pthread_mutex_un?
lock() is undefined.

RETURN VALUE

If successful, the pthread_mutex_lock(), pthread_mutex_trylock(), and pthread_mutex_unlock() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The pthread_mutex_lock() and pthread_mutex_trylock() functions shall fail if:

EAGAIN The mutex could not be acquired because the maximum number of recursive locks for mutex has been exceeded.

EINVAL The mutex was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex's current priority ceiling.

ENOTRECOVERABLE

The state protected by the mutex is not recoverable.

EOWNERDEAD

The mutex is a robust mutex and the process containing the pre? vious owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

The pthread_mutex_lock() function shall fail if:

EDEADLK

The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current thread already owns the mutex.

The pthread_mutex_trylock() function shall fail if:

EBUSY The mutex could not be acquired because it was already locked.

The pthread_mutex_unlock() function shall fail if:

EPERM The mutex type is PTHREAD_MUTEX_ERRORCHECK or PTHREAD_MUTEX_RE? CURSIVE, or the mutex is a robust mutex, and the current thread does not own the mutex.

The pthread_mutex_lock() and pthread_mutex_trylock() functions may fail if:

EOWNERDEAD

The mutex is a robust mutex and the previous owning thread ter? minated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

The pthread_mutex_lock() function may fail if:

EDEADLK

A deadlock condition was detected.

These functions shall not return an error code of [EINTR].

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently incon? sistent state is [EOWNERDEAD]. Applications that do not check the er? ror returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE

Mutex objects are intended to serve as a low-level primitive from which other thread synchronization functions can be built. As such, the im? plementation of mutexes should be as efficient as possible, and this has ramifications on the features available at the interface.

The mutex functions and the particular default settings of the mutex attributes have been motivated by the desire to not preclude fast, in? lined implementations of mutex locking and unlocking.

Since most attributes only need to be checked when a thread is going to be blocked, the use of attributes does not slow the (common) mutex-locking case.

Likewise, while being able to extract the thread ID of the owner of a

mutex might be desirable, it would require storing the current thread ID when each mutex is locked, and this could incur unacceptable levels of overhead. Similar arguments apply to a mutex_tryunlock operation.

For further rationale on the extended mutex types, see the Rationale (Informative) volume of POSIX.1?2017, Threads Extensions.

If an implementation detects that the value specified by the mutex ar? gument does not refer to an initialized mutex object, it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_consistent(), pthread_mutex_destroy(), pthread_mu? tex_timedlock(), pthread_mutexattr_getrobust()

The Base Definitions volume of POSIX.1?2017, Section 4.12, Memory Syn? chronization, <pthread.h>

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Por? table Operating System Interface (POSIX), The Open Group Base Specifi? cations Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and 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 .