**pthread\_mutex\_lock(3) - Linux man page**

**from** [**https://linux.die.net/man/3/pthread\_mutex\_lock**](https://linux.die.net/man/3/pthread_mutex_lock)

**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](https://linux.die.net/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 calling *pthread\_mutex\_lock*(). If the mutex is already locked, the calling thread shall block until 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 the mutex type is PTHREAD\_MUTEX\_NORMAL, deadlock detection shall not be provided. Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, undefined behavior results.

If the mutex type is PTHREAD\_MUTEX\_ERRORCHECK, then error checking shall be provided. If a thread attempts to relock a mutex that it has already locked, an error shall be returned. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error shall be returned.

If the mutex type is PTHREAD\_MUTEX\_RECURSIVE, then the mutex shall maintain the concept of a lock count. When a thread successfully acquires a mutex 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. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error shall be returned.

If the mutex type is PTHREAD\_MUTEX\_DEFAULT, attempting to recursively lock the mutex results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling thread results in undefined behavior. Attempting to unlock the mutex if it is not locked results in undefined behavior.

The *pthread\_mutex\_trylock*() function shall be equivalent to *pthread\_mutex\_lock*(), except that if the mutex object referenced by *mutex* is currently 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*() function shall immediately return success.

The *pthread\_mutex\_unlock*() function shall release the mutex object referenced 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.

**Return Value**

If successful, the *pthread\_mutex\_lock*() and *pthread\_mutex\_unlock*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

The *pthread\_mutex\_trylock*() function shall return zero if a lock on the mutex object referenced by *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

**Errors**

The *pthread\_mutex\_lock*() and *pthread\_mutex\_trylock*() functions shall fail if:

**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.

The *pthread\_mutex\_trylock*() function shall fail if:

**EBUSY**

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

The *pthread\_mutex\_lock*(), *pthread\_mutex\_trylock*(), and *pthread\_mutex\_unlock*() functions may fail if:

**EINVAL**

The value specified by *mutex* does not refer to an initialized mutex object.

**EAGAIN**

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

The *pthread\_mutex\_lock*() function may fail if:

**EDEADLK**

The current thread already owns the mutex.

The *pthread\_mutex\_unlock*() function may fail if:

**EPERM**

The current thread does not own the mutex.

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

*The following sections are informative.*

**Examples**

None.

**Application Usage**

None.

**Rationale**

Mutex objects are intended to serve as a low-level primitive from which other thread synchronization functions can be built. As such, the implementation 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, inlined implementations of mutex locking and unlocking.

For example, deadlocking on a double-lock is explicitly allowed behavior in order to avoid requiring more overhead in the basic mechanism than is absolutely necessary. (More "friendly" mutexes that detect deadlock or that allow multiple locking by the same thread are easily constructed by the user via the other mechanisms provided. For example, *pthread\_self*() can be used to record mutex ownership.) Implementations might also choose to provide such extended features as options via special mutex attributes.

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.

**Future Directions**

None.

**See Also**

*pthread\_mutex\_destroy*(), *pthread\_mutex\_timedlock*(), the Base Definitions volume of IEEE Std 1003.1-2001, *<[pthread.h](https://linux.die.net/include/pthread.h)>*

**Copyright**

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1, 2003 Edition, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright Â© 2001-2003 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> .

**Referenced By**

[**pthreads**](https://linux.die.net/man/7/pthreads)(7)