

**NAME**

**pthread\_mutexattr\_init**, **pthread\_mutexattr\_destroy**, **pthread\_mutexattr\_setprioceiling**,  
**pthread\_mutexattr\_getprioceiling**, **pthread\_mutexattr\_setprotocol**, **pthread\_mutexattr\_getprotocol**,  
**pthread\_mutexattr\_setpshared**, **pthread\_mutexattr\_getpshared**, **pthread\_mutexattr\_setrobust**,  
**pthread\_mutexattr\_getrobust**, **pthread\_mutexattr\_settype**, **pthread\_mutexattr\_gettype** - mutex attribute  
operations

**LIBRARY**

POSIX Threads Library (libpthread, -lpthread)

**SYNOPSIS**

```
#include <pthread.h>
```

*int*

```
pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

*int*

```
pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
```

*int*

```
pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr, int prioceiling);
```

*int*

```
pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *attr, int *prioceiling);
```

*int*

```
pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr, int protocol);
```

*int*

```
pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict attr, int *restrict protocol);
```

*int*

```
pthread_mutexattr_setpshared(pthread_mutexattr_t *attr, int shared);
```

*int*

```
pthread_mutexattr_getpshared(const pthread_mutexattr_t *attr, int *shared);
```

*int*

```
pthread_mutexattr_setrobust(pthread_mutexattr_t *attr, int robust);
```

*int*

**pthread\_mutexattr\_getrobust**(*pthread\_mutexattr\_t* \*attr, *int* \*robust);

*int*

**pthread\_mutexattr\_settype**(*pthread\_mutexattr\_t* \*attr, *int* type);

*int*

**pthread\_mutexattr\_gettype**(*const pthread\_mutexattr\_t* \*restrict attr, *int* \*restrict type);

## DESCRIPTION

Mutex attributes are used to specify parameters to **pthread\_mutex\_init**(). One attribute object can be used in multiple calls to **pthread\_mutex\_init**(), with or without modifications between calls.

The **pthread\_mutexattr\_init**() function initializes *attr* with all the default mutex attributes.

The **pthread\_mutexattr\_destroy**() function destroys *attr*.

The **pthread\_mutexattr\_setprioceiling**() function sets the priority ceiling for the mutex, used by threads executed under the PTHREAD\_PRIO\_PROTECT protocol.

The **pthread\_mutexattr\_setprotocol**() function specifies the protocol to be followed in utilizing mutexes. The *protocol* argument can take one of the following values:

PTHREAD\_PRIO\_NONE Priority and scheduling of the thread owning this mutex is not affected by its mutex ownership.

PTHREAD\_PRIO\_INHERIT Request priority-inheritance protocol, where the thread owning the mutex is executed at the highest priority among priorities of all threads waiting on any mutex owned by this thread.

PTHREAD\_PRIO\_PROTECT Request priority-inheritance protocol, where the thread owning the mutex is executed at highest priority among priorities or priority ceilings of all threads waiting on any mutex owned by this thread.

The **pthread\_mutexattr\_setpshared**() function sets the process-shared attribute of *attr* to the value specified in *pshared*. The argument *pshared* may have one of the following values:

PTHREAD\_PROCESS\_PRIVATE The mutex may only be used by threads in the same process as the one that created the object.

**PTHREAD\_PROCESS\_SHARED** The mutex may be used by threads in processes other than the one that created the object, assuming other processes share access to the memory where the mutex was allocated.

See `libthr(3)` for details of the implementation of the shared mutexes, and their limitations.

The **`pthread_mutexattr_setrobust()`** function specifies robustness attribute of the mutex. Possible values for the *robust* argument are

**PTHREAD\_MUTEX\_STALLED** No special actions are taken if the thread owning the mutex is terminated without unlocking the mutex lock. This can lead to deadlocks if no other thread can unlock the mutex. This is the default value.

**PTHREAD\_MUTEX\_ROBUST** If the process containing the owning thread of a robust mutex, or owning thread, terminates while holding the mutex lock, the next thread that acquires the mutex is notified about the termination by the return value `EOWNERDEAD` from the locking function. Then, either `pthread_mutex_consistent(3)` can be used to repair the mutex lock state, or `pthread_mutex_unlock(3)` can unlock the mutex lock but also put it an unusable state, where all further attempts to acquire it result in the `ENOTRECOVERABLE` error.

The **`pthread_mutexattr_settype()`** function sets the type of the mutex. The type affects the behavior of calls which lock and unlock the mutex. The possible values for the *type* argument are

**PTHREAD\_MUTEX\_NORMAL** Both recursive locking, and unlocking when the lock is not owned by the current thread, cause an error to be returned from the corresponding functions. This matches `PTHREAD_MUTEX_ERRORCHECK` but somewhat contradicts the behavior mandated by POSIX.

**PTHREAD\_MUTEX\_ERRORCHECK** Both recursive locking, and unlocking when the lock is not owned by the current thread, cause an error returned from the corresponding functions.

**PTHREAD\_MUTEX\_RECURSIVE** Recursive locking is allowed. Attempt to unlock when current thread is not an owner of the lock causes an error to be returned.

**PTHREAD\_MUTEX\_DEFAULT** The FreeBSD implementation maps this type to

PTHREAD\_MUTEX\_ERRORCHECK type.

The **pthread\_mutexattr\_get\***(*n*) functions copy the value of the attribute that corresponds to each function name to the location pointed to by the second function parameter.

## RETURN VALUES

If successful, these functions return 0. Otherwise, an error number is returned to indicate the error.

## ERRORS

The **pthread\_mutexattr\_init**(*n*) function will fail if:

[ENOMEM]           Out of memory.

The **pthread\_mutexattr\_destroy**(*n*) function will fail if:

[EINVAL]           Invalid value for *attr*.

The **pthread\_mutexattr\_setprioceiling**(*n*, *prioceiling*) function will fail if:

[EINVAL]           Invalid value for *attr*, or invalid value for *prioceiling*.

The **pthread\_mutexattr\_getprioceiling**(*n*, *prioceiling*) function will fail if:

[EINVAL]           Invalid value for *attr*.

The **pthread\_mutexattr\_setprotocol**(*n*, *protocol*) function will fail if:

[EINVAL]           Invalid value for *attr*, or invalid value for *protocol*.

The **pthread\_mutexattr\_getprotocol**(*n*, *protocol*) function will fail if:

[EINVAL]           Invalid value for *attr*.

The **pthread\_mutexattr\_setpshared**(*n*, *shared*) function will fail if:

[EINVAL]           Invalid value for *attr*, or invalid value for *shared*.

The **pthread\_mutexattr\_getpshared**(*n*, *shared*) function will fail if:

[EINVAL]           Invalid value for *attr*.

The **pthread\_mutexattr\_settype()** function will fail if:

[EINVAL] Invalid value for *attr*, or invalid value for *type*.

The **pthread\_mutexattr\_gettype()** function will fail if:

[EINVAL] Invalid value for *attr*.

The **pthread\_mutexattr\_setrobust()** function will fail if:

[EINVAL] Invalid value for *attr*, or invalid value for *robust*.

The **pthread\_mutexattr\_getrobust()** function will fail if:

[EINVAL] Invalid value for *attr*.

## SEE ALSO

libthr(3), pthread\_mutex\_init(3)

## STANDARDS

The **pthread\_mutexattr\_init()** and **pthread\_mutexattr\_destroy()** functions conform to ISO/IEC 9945-1:1996 ("POSIX.1")

The **pthread\_mutexattr\_setprioceiling()**, **pthread\_mutexattr\_getprioceiling()**, **pthread\_mutexattr\_setprotocol()**, **pthread\_mutexattr\_getprotocol()**, **pthread\_mutexattr\_setpshared()**, **pthread\_mutexattr\_getpshared()**, **pthread\_mutexattr\_settype()**, and **pthread\_mutexattr\_gettype()** functions conform to Version 2 of the Single UNIX Specification ("SUSv2"). The **pthread\_mutexattr\_setrobust()** and **pthread\_mutexattr\_getrobust()** functions conform to Version 4 of the Single UNIX Specification ("SUSv4").