Node:Mutexes, Next:Condition Variables, Previous:Cleanup Handlers, Up:POSIX Threads

---

Mutexes

A mutex is a MUTual EXclusion device, and is useful for protecting shared data structures from concurrent modifications, and implementing critical sections and monitors.

A mutex has two possible states: unlocked (not owned by any thread), and locked (owned by one thread). A mutex can never be owned by two different threads simultaneously. A thread attempting to lock a mutex that is already locked by another thread is suspended until the owning thread unlocks the mutex first.

None of the mutex functions is a cancellation point, not even `pthread_mutex_lock`, in spite of the fact that it can suspend a thread for arbitrary durations. This way, the status of mutexes at cancellation points is predictable, allowing cancellation handlers to unlock precisely those mutexes that need to be unlocked before the thread stops executing. Consequently, threads using deferred cancellation should never hold a mutex for extended periods of time.

It is not safe to call mutex functions from a signal handler. In particular, calling `pthread_mutex_lock` or `pthread_mutex_unlock` from a signal handler may deadlock the calling thread.

int **pthread_mutex_init** (*pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr*) Function

`pthread_mutex_init` initializes the mutex object pointed to by *mutex* according to the mutex attributes specified in *mutexattr*. If *mutexattr* is `NULL`, default attributes are used instead.

The LinuxThreads implementation supports only one mutex attribute, the *mutex type*, which is either "fast", "recursive", or "error checking". The type of a mutex determines whether it can be locked again by a thread that already owns it. The default type is "fast".

Variables of type `pthread_mutex_t` can also be initialized statically, using the constants `PTHREAD_MUTEX_INITIALIZER` (for timed mutexes), `PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP` (for recursive mutexes), `PTHREAD_ADAPTIVE_MUTEX_INITIALIZER_NP` (for fast mutexes(, and `PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP` (for error checking mutexes).

`pthread_mutex_init` always returns 0.

int **pthread_mutex_lock** (*pthread_mutex_t *mutex*)) Function

`pthread_mutex_lock` locks the given mutex. If the mutex is currently unlocked, it becomes locked and owned by the calling thread, and `pthread_mutex_lock` returns immediately. If the mutex is already locked by another thread, `pthread_mutex_lock` suspends the calling thread until the mutex is unlocked.

If the mutex is already locked by the calling thread, the behavior of `pthread_mutex_lock` depends on the type of the mutex. If the mutex is of the "fast" type, the calling thread is suspended. It will remain suspended forever, because no other thread can unlock the mutex. If the mutex is of the "error checking" type, `pthread_mutex_lock` returns immediately with the error code `EDEADLK`. If the mutex is of the "recursive" type, `pthread_mutex_lock` succeeds and returns immediately, recording the number of times the calling thread has locked the mutex. An equal number of `pthread_mutex_unlock` operations must be performed before the mutex returns to the unlocked

state.

int **pthread_mutex_trylock** (*pthread_mutex_t *mutex*) Function

`pthread_mutex_trylock` behaves identically to `pthread_mutex_lock`, except that it does not block the calling thread if the mutex is already locked by another thread (or by the calling thread in the case of a "fast" mutex). Instead, `pthread_mutex_trylock` returns immediately with the error code `EBUSY`.

int **pthread_mutex_timedlock** (*pthread_mutex_t *mutex, const struct timespec *abstime*) Function

The `pthread_mutex_timedlock` is similar to the `pthread_mutex_lock` function but instead of blocking for in indefinite time if the mutex is locked by another thread, it returns when the time specified in *abstime* is reached.

This function can only be used on standard ("timed") and "error checking" mutexes. It behaves just like `pthread_mutex_lock` for all other types.

If the mutex is successfully locked, the function returns zero. If the time specified in *abstime* is reached without the mutex being locked, `ETIMEDOUT` is returned.

This function was introduced in the POSIX.1d revision of the POSIX standard.

int **pthread_mutex_unlock** (*pthread_mutex_t *mutex*) Function

`pthread_mutex_unlock` unlocks the given mutex. The mutex is assumed to be locked and owned by the calling thread on entrance to `pthread_mutex_unlock`. If the mutex is of the "fast" type, `pthread_mutex_unlock` always returns it to the unlocked state. If it is of the "recursive" type, it decrements the locking count of the mutex (number of `pthread_mutex_lock` operations performed on it by the calling thread), and only when this count reaches zero is the mutex actually unlocked.

On "error checking" mutexes, `pthread_mutex_unlock` actually checks at run-time that the mutex is locked on entrance, and that it was locked by the same thread that is now calling `pthread_mutex_unlock`. If these conditions are not met, `pthread_mutex_unlock` returns `EPERM`, and the mutex remains unchanged. "Fast" and "recursive" mutexes perform no such checks, thus allowing a locked mutex to be unlocked by a thread other than its owner. This is non-portable behavior and must not be relied upon.

int **pthread_mutex_destroy** (*pthread_mutex_t *mutex*) Function

`pthread_mutex_destroy` destroys a mutex object, freeing the resources it might hold. The mutex must be unlocked on entrance. In the LinuxThreads implementation, no resources are associated with mutex objects, thus `pthread_mutex_destroy` actually does nothing except checking that the mutex is unlocked.

If the mutex is locked by some thread, `pthread_mutex_destroy` returns `EBUSY`. Otherwise it returns 0.

If any of the above functions (except `pthread_mutex_init`) is applied to an uninitialized mutex, they will simply return `EINVAL` and do nothing.

A shared global variable *x* can be protected by a mutex as follows:

```
int x;
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
```

All accesses and modifications to *x* should be bracketed by calls to `pthread_mutex_lock` and `pthread_mutex_unlock` as follows:

```
pthread_mutex_lock(&mut);
/* operate on x */
pthread_mutex_unlock(&mut);
```

Mutex attributes can be specified at mutex creation time, by passing a mutex attribute object as second argument to `pthread_mutex_init`. Passing `NULL` is equivalent to passing a mutex attribute object with all attributes set to their default values.

int **pthread_mutexattr_init** (*pthread_mutexattr_t *attr*) Function

`pthread_mutexattr_init` initializes the mutex attribute object *attr* and fills it with default values for the attributes.

This function always returns 0.

int **pthread_mutexattr_destroy** (*pthread_mutexattr_t *attr*) Function

`pthread_mutexattr_destroy` destroys a mutex attribute object, which must not be reused until it is reinitialized. `pthread_mutexattr_destroy` does nothing in the LinuxThreads implementation.

This function always returns 0.

LinuxThreads supports only one mutex attribute: the mutex type, which is either `PTHREAD_MUTEX_ADAPTIVE_NP` for "fast" mutexes, `PTHREAD_MUTEX_RECURSIVE_NP` for "recursive" mutexes, `PTHREAD_MUTEX_TIMED_NP` for "timed" mutexes, or `PTHREAD_MUTEX_ERRORCHECK_NP` for "error checking" mutexes. As the `NP` suffix indicates, this is a non-portable extension to the POSIX standard and should not be employed in portable programs.

The mutex type determines what happens if a thread attempts to lock a mutex it already owns with `pthread_mutex_lock`. If the mutex is of the "fast" type, `pthread_mutex_lock` simply suspends the calling thread forever. If the mutex is of the "error checking" type, `pthread_mutex_lock` returns immediately with the error code `EDEADLK`. If the mutex is of the "recursive" type, the call to `pthread_mutex_lock` returns immediately with a success return code. The number of times the thread owning the mutex has locked it is recorded in the mutex. The owning thread must call `pthread_mutex_unlock` the same number of times before the mutex returns to the unlocked state.

The default mutex type is "timed", that is, `PTHREAD_MUTEX_TIMED_NP`.

int **pthread_mutexattr_settype** (*pthread_mutexattr_t *attr, int type*) Function

`pthread_mutexattr_settype` sets the mutex type attribute in *attr* to the value specified by *type*.

If *type* is not `PTHREAD_MUTEX_ADAPTIVE_NP`, `PTHREAD_MUTEX_RECURSIVE_NP`, `PTHREAD_MUTEX_TIMED_NP`, or `PTHREAD_MUTEX_ERRORCHECK_NP`, this function will return `EINVAL` and leave *attr* unchanged.

The standard Unix98 identifiers `PTHREAD_MUTEX_DEFAULT`, `PTHREAD_MUTEX_NORMAL`, `PTHREAD_MUTEX_RECURSIVE`, and `PTHREAD_MUTEX_ERRORCHECK` are also permitted.

int **pthread_mutexattr_gettype** (*const pthread_mutexattr_t *attr, int *type*) Function

`pthread_mutexattr_gettype` retrieves the current value of the mutex type attribute in *attr* and

stores it in the location pointed to by *type*.

This function always returns 0.