Using Read-Write Locks

After the attributes for a read-write lock are configured, you initialize the read-write lock itself. The following functions are used to initialize or destroy, lock or unlock, or try to lock a read-write lock. The following table lists the functions discussed in this section that manipulate read-write locks.

Table 4-9 Routines that Manipulate Read-Write Locks

Operation	Destination Discussion
Initialize a read-write lock	"pthread_rwlock_init(3THR)"
Read lock on read-write lock	"pthread_rwlock_rdlock(3THR)"
Read lock with a nonblocking read-write lock	"pthread_rwlock_tryrdlock(3THR)"
Write lock on read-write lock	"pthread_rwlock_wrlock(3THR)"
Write lock with a nonblocking read-write lock	"pthread_rwlock_trywrlock(3THR)"
Unlock a read-write lock	"pthread_rwlock_unlock(3THR)"
Destroy a read-write lock	"pthread_rwlock_destroy(3THR)"

Initialize a Read-Write Lock

pthread_rwlock_init(3THR)

```
#include <pthread.h>

int pthread_rwlock_init(pthread_rwlock_t *rwlock, const pthread_rwlockattr_t
*attr);

pthread_rwlock_t  rwlock = PTHREAD_RWLOCK_INITIALIZER;
```

Use pthread_rwlock_init(3THR) to initialize the read-write lock referenced by **rwlock** with the attributes referenced by **attr**. If **attr** is `NULL`, the default read-write lock attributes are used; the effect is the same as passing the address of a default read-write lock attributes object. Once initialized, the lock can be used any number of times without being re-initialized. On successful initialization, the state of the read-write lock becomes initialized and unlocked. Results are undefined if **`pthread_rwlock_init()`** is called specifying an already initialized read-write lock. Results are undefined if a read-write lock is used without first being initialized. (For Solaris threads, see "rwlock_init(3THR)".)

In cases where default read-write lock attributes are appropriate, the macro `PTHREAD_RWLOCK_INITIALIZER` can be used to initialize read-write locks that are statically allocated. The effect is equivalent to dynamic initialization by a call to **`pthread_rwlock_init()`** with the parameter **attr** specified as `NULL`, except that no error checks are performed.

Return Value

If successful, **`pthread_rwlock_init()`** returns zero. Otherwise, an error number is returned to indicate the error.

If **`pthread_rwlock_init()`** fails, **rwlock** is not initialized and the contents of **rwlock** are undefined.

`EINVAL`

The value specified by **attr** or **rwlock** is invalid.

Read Lock on Read-Write Lock

pthread_rwlock_rdlock(3THR)

```
#include <pthread.h>

int  pthread_rwlock_rdlock(pthread_rwlock_t *rwlock );
```

pthread_rwlock_rdlock(3THR) applies a read lock to the read-write lock referenced by **rwlock**. The calling thread acquires the read lock if a writer does not hold the lock and there are no writers blocked on the lock. It is unspecified whether the calling thread acquires the lock when a writer does not hold the lock and there are writers waiting for the lock. If a writer holds the lock, the calling thread will not acquire the read lock. If the read lock is not acquired, the calling thread blocks (that is, it does not return from the **`pthread_rwlock_rdlock()`**) until it can acquire the lock. Results are undefined if the calling thread holds a write lock on **rwlock** at the time the call is made.

Implementations are allowed to favor writers over readers to avoid writer starvation. (For instance, the Solaris threads implementation favors writers over readers. See "rw_rdlock(3THR)".)

A thread can hold multiple concurrent read locks on **rwlock** (that is, successfully call **`pthread_rwlock_rdlock()`** n times) If so, the thread must perform matching unlocks (that is, it must call **`pthread_rwlock_unlock()`** **n** times).

Results are undefined if **`pthread_rwlock_rdlock()`** is called with an uninitialized read-write lock.

If a signal is delivered to a thread waiting for a read-write lock for reading, on return from the signal handler the thread resumes waiting for the read-write lock for reading as if it was not interrupted.

Return Value

If successful, **`pthread_rwlock_rdlock()`** returns zero. Otherwise, an error number is returned to indicate the error.

`EINVAL`

The value specified by **attr** or **rwlock** is invalid.

Read Lock With a Nonblocking Read-Write Lock

pthread_rwlock_tryrdlock(3THR)

```
#include <pthread.h>

int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

pthread_rwlock_tryrdlock(3THR) applies a read lock as in **`pthread_rwlock_rdlock()`** with the exception that the function fails if any thread holds a write lock on **rwlock** or there are writers blocked on **rwlock**. (For Solaris threads, see "rw_tryrdlock(3THR)".)

Return Value

`pthread_rwlock_tryrdlock()` returns zero if the lock for reading on the read-write lock object referenced by **rwlock** is acquired. Otherwise an error number is returned to indicate the error.

`EBUSY`

The read-write lock could not be acquired for reading because a writer holds the lock or was blocked on it.

Write Lock on Read-Write Lock

pthread_rwlock_wrlock(3THR)

```
#include <pthread.h>

int  pthread_rwlock_wrlock(pthread_rwlock_t *rwlock );
```

pthread_rwlock_wrlock(3THR) applies a write lock to the read-write lock referenced by **rwlock**. The calling thread acquires the write lock if no other thread (reader or writer) holds the read-write lock **rwlock**. Otherwise, the thread blocks (that is, does not return from the **`pthread_rwlock_wrlock()`** call) until it can acquire the lock. Results are undefined if the calling thread holds the read-write lock (whether a read or write lock) at the time the call is made.

Implementations are allowed to favor writers over readers to avoid writer starvation. (For instance, the Solaris threads implementation favors writers over readers. See "rw_wrlock(3THR)".)

Results are undefined if **`pthread_rwlock_wrlock()`** is called with an uninitialized read-write lock.

If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the signal handler the thread resumes waiting for the read-write lock for writing as if it was not interrupted.

Return Value

`pthread_rwlock_rwlock()` returns zero if the lock for writing on the read-write lock object referenced by **rwlock** is acquired. Otherwise an error number is returned to indicate the error.

Write Lock With a Nonblocking Read-Write Lock

pthread_rwlock_trywrlock(3THR)

```
#include <pthread.h>

int pthread_rwlock_trywrlock(pthread_rwlock_t  *rwlock);
```

pthread_rwlock_trywrlock(3THR) applies a write lock like **`pthread_rwlock_wrlock()`**, with the exception that the function fails if any thread currently holds **rwlock** (for reading or writing). (For Solaris threads, see "rw_trywrlock(3THR)".)

Results are undefined if **`pthread_rwlock_trywrlock()`** is called with an uninitialized read-write lock.

If a signal is delivered to a thread waiting for a read-write lock for writing, on return from the signal handler the thread resumes waiting for the read-write lock for writing as if it was not interrupted.

Return Value

If successful, **pthread_rwlock_trywrlock()** returns zero if the lock for writing on the read-write lock object referenced by **rwlock** is acquired. Otherwise, an error number is returned to indicate the error.

EBUSY

The read-write lock could not be acquired for writing because it is already locked for reading or writing.

Unlock a Read-Write Lock

pthread_rwlock_unlock(3THR)

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

pthread_rwlock_unlock(3THR) releases a lock held on the read-write lock object referenced by **rwlock**. Results are undefined if the read-write lock **rwlock** is not held by the calling thread. (For Solaris threads, see "rw_unlock(3THR)".)

If `pthread_rwlock_unlock` is called to release a read lock from the read-write lock object and there are other read locks currently held on this read-write lock object, the read-write lock object remains in the read locked state. If **pthread_rwlock_unlock()** releases the calling thread's last read lock on this read-write lock object, then the calling thread is no longer one of the owners of the object. If **pthread_rwlock_unlock()** releases the last read lock for this read-write lock object, the read-write lock object will be put in the unlocked state with no owners.

If **pthread_rwlock_unlock()** is called to release a write lock for this read-write lock object, the read-write lock object will be put in the unlocked state with no owners.

If the call to the **pthread_rwlock_unlock()** results in the read-write lock object becoming unlocked and there are multiple threads waiting to acquire the read-write lock object for writing, the scheduling policy is used to determine which thread acquires the read-write lock object for writing. If there are multiple threads waiting to acquire the read-write lock object for reading, the scheduling policy is used to determine the order in which the waiting threads acquire the read-write lock object for reading. If there are multiple threads blocked on **rwlock** for both read locks and write locks, it is unspecified whether the readers acquire the lock first or whether a writer acquires the lock first.

Results are undefined if **pthread_rwlock_unlock()** is called with an uninitialized read-write lock.

Return Value

If successful, **`pthread_rwlock_unlock()`** returns zero. Otherwise, an error number is returned to indicate the error.

Destroy a Read-Write Lock

pthread_rwlock_destroy(3THR)

```
#include <pthread.h>

int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);

pthread_rwlock_t  rwlock = PTHREAD_RWLOCK_INITIALIZER;
```

pthread_rwlock_destroy(3THR) destroys the read-write lock object referenced by **rwlock** and releases any resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is re-initialized by another call to **`pthread_rwlock_init()`** An implementation can cause **`pthread_rwlock_destroy()`** to set the object referenced by **rwlock** to an invalid value. Results are undefined if **`pthread_rwlock_destroy()`** is called when any thread holds **rwlock**. Attempting to destroy an uninitialized read-write lock results in undefined behavior. A destroyed read-write lock object can be re-initialized using **`pthread_rwlock_init()`**; the results of otherwise referencing the read-write lock object after it has been destroyed are undefined. (For Solaris threads, see "rwlock_destroy(3THR)".)

Return Value

If successful, **`pthread_rwlock_destroy()`** returns zero. Otherwise, an error number is returned to indicate the error.

`EINVAL`

The value specified by **attr** or **rwlock** is invalid