NAME

rwlock, rwlock_init, rwlock_destroy, rw_rdlock, rw_wrlock, rw_tryrdlock, rw_trywrlock, rw_unlock – multiple readers, single writer locks

SYNOPSIS

cc [*flag* ...] *file* ... **–lthread –lc** [*library* ...]

#include <synch.h>

int rwlock_init(rwlock_t ∗*rwlp*, **int** *type*, **void** ∗ *arg***);**

int rwlock_destroy(rwlock_t ∗*rwlp***);**

int rw_rdlock(rwlock_t ∗*rwlp***);**

int rw_wrlock(rwlock_t ∗*rwlp***);**

int rw_unlock(rwlock_t ∗*rwlp***);**

int rw_tryrdlock(rwlock_t ∗*rwlp***);**

int rw_trywrlock(rwlock_t ∗*rwlp***);**

MT-LEVEL

MT-Safe

DESCRIPTION

Many threads can have simultaneous read-only access to data, while only one thread can have write access at any given time. Multiple read access with single write access is controlled by locks, which are generally used to protect data that is frequently searched.

Readers/writer locks can synchronize threads in this process and other processes if they are allocated in writable memory and shared among cooperating processes (see **mmap**(2)), and are initialized for this purpose.

Additionally, readers/writer locks must be initialized prior to use. **rwlock_init( )** The readers/writer lock pointed to by *rwlp* is initialized by **rwlock_init( )**. A readers/writer lock is capable of having several types of behavior, which is specified by *type*. *arg* is currently not used, although a future type may define new behavior parameters via *arg*.

type may be one of the following:

USYNC_PROCESS The readers/writer lock can synchronize threads in this process and other processes. The readers/writer lock should be initialized by only one lock. *arg* is ignored.

USYNC_THREAD The readers/writer lock can synchronize threads in this process, only. *arg* is ignored.

Additionally, readers/writer locks can be initialized by allocation in zeroed memory. A *type* of **USYNC_THREAD** is assumed in this case. Multiple threads must not simultaneously initialize the same readers/writer lock. And a readers/writer lock must not be re-initialized while in use by other threads.

The following are default readers/writer lock initialization (intra-process):

```
rwlock_t rwlp;

rwlock_init(&rwlp, NULL, NULL);
```

OR

rwlock_init(&rwlp, USYNC_THREAD, NULL);

OR

rwlock_t rwlp = DEFAULTRWLOCK;

The following is a customized readers/writer lock initialization (inter-process):

rwlock_init(&rwlp, USYNC_PROCESS, NULL);

Any state associated with the readers/writer lock pointed to by *rwlp* are destroyed by **rwlock_destroy( )** and the readers/writer lock storage space is not released.

rw_rdlock( ) gets a read lock on the readers/writer lock pointed to by *rwlp*. If the readers/writer lock is currently locked for writing, the calling thread blocks until the write lock is freed. Multiple threads may simultaneously hold a read lock on a readers/writer lock.

rw_tryrdlock( ) trys to get a read lock on the readers/writer lock pointed to by *rwlp*. If the readers/writer lock is locked for writing, it returns an error; otherwise, the read lock is acquired.

rw_wrlock( ) gets a write lock on the readers/writer lock pointed to by *rwlp*. If the readers/writer lock is currently locked for reading or writing, the calling thread blocks until all the read and write locks are freed. At any given time, only one thread may have a write lock on a readers/writer lock.

rw_trywrlock( ) trys to get a write lock on the readers/writer lock pointed to by *rwlp*. If the readers/writer lock is currently locked for reading or writing, it returns an error.

rw_unlock( ) unlocks a readers/writer lock pointed to by *rwlp*, if the readers/writer lock is locked and the calling thread holds the lock for either reading or writing. One of the other threads that is waiting for the readers/writer lock to be freed will be unblocked, provided there is other waiting threads. If the calling thread does not hold the lock for either reading or writing, no error status is returned, and the program's behavior is unknown.

RETURN VALUES

Upon successful completion, **0** is returned; otherwise, a non-zero value indicates an error.

ERRORS

These functions fail and return the corresponding value if any of the following conditions are detected.

EINVAL Invalid argument.

EFAULT *rwlp* or *arg* point to an illegal address.

rw_tryrdlock( ) or **rw_trywrlock( )** fails and returns the corresponding value if any of the following conditions are detected.

EBUSY The readers/writer lock pointed to by *rwlp* was already locked.

SEE ALSO

mmap(2)

NOTES

These interfaces also available via:

#include <thread.h>

If multiple threads are waiting for a readers/writer lock, the acquisition order is random by default. However, some implementations may bias acquisition order to avoid depriving writers. The current implementation favors writers over readers.