**NAME**

rwlock, rw_init, rw_destroy, rw_enter, rw_exit, rw_tryenter, rw_downgrade, rw_tryupgrade, rw_read_locked – readers/writer lock functions

**SYNOPSIS**

**#include <sys/ksynch.h>**

**void rw_init(krwlock_t** **rwlp*, **char** **name*, **krw_type_t** *type,* **void** **arg*);

**void rw_destroy(krwlock_t** **rwlp*);

**void rw_enter(krwlock_t** **rwlp*, **krw_t** *enter_type*);

**void rw_exit(krwlock_t** **rwlp*);

**int rw_tryenter(krwlock_t** **rwlp*, **krw_t** *enter_type*);

**void rw_downgrade(krwlock_t** **rwlp*);

**int rw_tryupgrade(krwlock_t** **rwlp*);

**int rw_read_locked(krwlock_t** **rwlp*);

**INTERFACE LEVEL**

Solaris DDI specific (Solaris DDI).

**ARGUMENTS**

| | |
|---|---|
| *rwlp* | Pointer to a **krwlock_t** readers/writer lock. |
| *name* | Character string describing lock for statistics and debugging. |
| *type* | Type of readers/writer lock. |
| *arg* | Type-specific argument for initialization function. |
| *enter_type* | Indication of whether the lock is to be acquired non-exclusively or exclusively **RW_READER** or **RW_WRITER**. |

**DESCRIPTION**

A multiple-readers, single-writer lock is represented by the **krwlock_t** data type. This type of lock will allow many threads to have simultaneous read-only access to an object. Only one thread may have write access at any one time. An object which is searched more frequently than it is changed is a good candidate for a readers/writer lock.

Readers/writer locks can be more than twice as expensive as a mutex lock, and the advantage of multiple read access may not occur if the lock will only be held for a short time.

**rw_init( )** initializes a readers/writer lock. It is an error to initialize a lock more than once. The *type* argument should be set to **RW_DRIVER**. If the lock is used by the interrupt handler, the type-specific argument, *arg,* should be the **ddi_iblock_cookie** returned from **ddi_get_iblock_cookie**(9F) or **ddi_get_soft_iblock_cookie**(9F). If the lock is not used by any interrupt handler, the argument should be **NULL**.

If the call to **rw_init( )** is compiled with **_LOCKTEST** or **_MPSTATS** defined, statistics will be kept for the lock. This may have a performance penalty.

**rw_destroy( )** releases any storage that might have been allocated by **rw_init( )**. It should be called before deallocating the storage containing the lock.

**rw_enter( )** acquires the lock, and blocks if necessary. If *enter_type* is **RW_READER**, the caller blocks if there is a writer or a thread attempting to enter for writing. If *enter_type* is **RW_WRITER**, the caller blocks if any thread holds the lock.

**rw_exit( )** releases the lock and may wake up one or more threads waiting on the lock.

**rw_tryenter( )** attempts to enter the lock, like **rw_enter( )**, but never blocks. It returns a non-zero value if the lock was successfully entered, and zero otherwise.

A thread which holds the lock exclusively (entered with **RW_WRITER** ), may call **rw_downgrade( )** to convert to holding the lock non-exclusively (as if entered with **RW_READER** ). Other waiting readers will be unblocked unless there is a waiting writer.

**rw_tryupgrade( )** can be called by a thread which holds the lock for reading to attempt to convert to holding it for writing. This upgrade can only succeed if no other thread is holding the lock and no other thread is blocked waiting to acquire the lock for writing.

**rw_read_locked( )** returns non-zero if the calling thread holds the lock for read, and zero if the caller holds the lock for write. The caller must hold the lock. The system may panic if **rw_read_locked( )** is called for a lock that isn't held by the caller.

## RETURN VALUES

| | |
|---|---|
| **0** | **rw_tryenter( )** could not obtain the lock without blocking. |
| **0** | **rw_tryupgrade( )** was unable to perform the upgrade because of other threads holding or waiting to hold the lock. |
| **0** | **rw_read_locked( )** returns **0** if the lock is held by the caller for write. |
| non-zero | from **rw_read_locked( )** if the lock is held by the caller for read. non-zero successful return from **rw_tryenter( )** or **rw_tryupgrade( )**. |

## CONTEXT

These functions can be called from user or interrupt context, except for **rw_init( )** and **rw_destroy( )**, which can be called from user context only.

## SEE ALSO

**condvar**(9F), **ddi_add_intr**(9F), **ddi_get_iblock_cookie**(9F), **mutex**(9F), **semaphore**(9F)

*Writing Device Drivers*