NAME

mutex, mutex_enter, mutex_exit, mutex_init, mutex_destroy, mutex_owned, mutex_tryenter – mutual exclusion lock routines

SYNOPSIS

#include <sys/ksynch.h>

void mutex_init(kmutex_t **mp,* **char** **name,* **kmutex_type_t** *type***, void** **arg***);**

void mutex_destroy(kmutex_t **mp***);**

void mutex_enter(kmutex_t **mp***);**

void mutex_exit(kmutex_t **mp***);**

int mutex_owned(kmutex_t **mp***);**

int mutex_tryenter(kmutex_t **mp***);**

INTERFACE LEVEL

Solaris DDI specific (Solaris DDI).

ARGUMENTS

mp	Pointer to a kernel mutex lock (**kmutex_t**).
name	Character string describing lock for statistics and debugging.
type	Type of mutex lock.
arg	Type-specific argument for initialization routine.

DESCRIPTION

A mutex enforces a policy of mutual exclusion. Only one thread at a time may hold a particular mutex. Threads trying to lock a held mutex will block until the mutex is unlocked.

Mutexes are strictly bracketing and may not be recursively locked. That is to say, mutexes should be exited in the opposite order they were entered, and cannot be reentered before exiting.

mutex_init( ) is used to initialize a mutex so that it is unlocked and has a particular variant type. The only DDI-compliant types provided are **MUTEX_DRIVER**, **MUTEX_DRIVER_NOSTAT**, and **MUTEX_DRIVER_STAT**. Most of the time, the type **MUTEX_DRIVER** should be used.

If the call is compiled with **_LOCKTEST** or **_MPSTATS** defined, statistics will be kept for **MUTEX_DRIVER** mutexes. Statistics are always maintained for type **MUTEX_DRIVER_STAT,** and never maintained for **MUTEX_DRIVER_NOSTAT**. Note that statistics may incur a performance penalty. In addition, the system may need to allocate memory associated with the mutex, depending on the type.

arg provides type-specific information for a given variant type of mutex. When **mutex_init( )** is called for driver mutexes, if the mutex is used by the interrupt handler, the arg should be the **ddi_iblock_cookie** returned from **ddi_get_iblock_cookie**(9F) or **ddi_get_soft_iblock_cookie**(9F). If the mutex is never used inside an interrupt handler, the argument should be **NULL**.

mutex_enter( ) is used to acquire a mutex. If the mutex is already held, then the caller blocks. After returning, the calling thread is the owner of the mutex. If the mutex is already held by the calling thread, a panic will ensue.

mutex_owned( ) should only be used in **ASSERT( )**s, and may be enforced by not being defined unless the preprocessor symbol **DEBUG** is defined. Its return value is non-zero if the current thread (or, if that cannot be determined, at least some thread) holds the mutex pointed to by *mp*.

mutex_tryenter( ) is very similar to **mutex_enter( )** except that it doesn't block when the mutex is already held. **mutex_tryenter( )** returns non-zero when it acquired the mutex and 0 when the mutex is already held.

mutex_exit( ) releases a mutex and will unblock another thread if any are blocked on the mutex.

mutex_destroy( ) frees any storage associated with the mutex, which may have been allocated when **mutex_init( )** was called. This should be called before deallocating storage containing the mutex. The caller must somehow be sure that no other thread will attempt to use the mutex.

RETURN VALUES

mutex_tryenter( ) returns non-zero on success and zero of failure.

mutex_owned( ) returns non-zero if the calling thread currently holds the mutex pointed to by *mp*, or when that cannot be determined, if any thread holds the mutex. **mutex_owned( )** returns zero otherwise.

CONTEXT

These functions can be called from user, kernel, or high-level interrupt context, except for **mutex_init( )** and **mutex_destroy( )**, which can be called from user or kernel context only.

EXAMPLES

Initialization

A driver might do this to initialize a mutex that is part of its unit structure and used in its interrupt routine:

```
ddi_get_iblock_cookie(dip, 0, &iblock);
mutex_init(&un->un_lock, "xx unit lock", MUTEX_DRIVER,
        (void *)iblock);
ddi_add_intr(dip, 0, NULL, &dev_cookie, xxintr,
        (caddr_t)un);
```

Also, a routine that expects to be called with a certain lock held might have the following ASSERT:

```
xxstart(struct xxunit *un)
{
        ASSERT(mutex_owned(&un->un_lock));
...
```

SEE ALSO

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

Writing Device Drivers

BUGS

There is currently no product support for looking at lock statistics.