NAME

semaphore, sema_init, sema_destroy, sema_p, sema_p_sig, sema_v, sema_tryp – semaphore functions

SYNOPSIS

#include <sys/ksynch.h>

void sema_init(ksema_t **sp*, **u_int** *val*, **char** **name*, **ksema_type_t** *type*, **void** **arg***);**

void sema_destroy(ksema_t **sp***);**

void sema_p(ksema_t **sp***);**

void sema_v(ksema_t **sp***);**

int sema_p_sig(ksema_t **sp***);**

int sema_tryp(ksema_t **sp***);**

INTERFACE LEVEL

Solaris DDI specific (Solaris DDI).

ARGUMENTS

sp A pointer to a semaphore, type **ksema_t**.

val Initial value for semaphore.

name A string describing the semaphore for statistics and debugging.

type Variant type of the semaphore. Currently only **SEMA_DRIVER** is supported.

arg Type-specific argument, should be **NULL**.

DESCRIPTION

These functions implement counting semaphores as described by Dijkstra. A semaphore has a value which is atomicly decremented by **sema_p( )** and atomicly incremented by **sema_v( )**. The value must always be greater than or equal to zero. If **sema_p( )** is called and the value is zero, the calling thread is blocked until another thread performs a **sema_v( )** operation on the semaphore.

Semaphores are initialized by calling **sema_init( )**. The argument, *val*, gives the initial value for the semaphore. The semaphore storage is provided by the caller but more may be dynamicly allocated, if necessary, by **sema_init( )**. For this reason, **sema_destroy( )** should be called before deallocating the storage containing the semaphore.

sema_p_sig( ) decrements the semaphore, as does **sema_p( )**, however, if the semaphore value is zero, **sema_p_sig( )** will return without decrementing the value if a signal (e.g. from **kill**(2)) is pending for the thread.

sema_tryp( ) will decrement the semaphore value only if it is greater than zero, and will not block.

RETURN VALUES

0 **sema_tryp( )** could not decrement the semaphore value because it was zero.

1 **sema_p_sig( )** was not able to decrement the semaphore value and detected a pending signal.

CONTEXT

These function can be called from user or interrupt context, except for **sema_init( )** and **sema_destroy( )**, which can be called from user context only. None of these functions can be called from a high-level interrupt context. In most cases, **sema_init( )** and **sema_destroy( )** should not be called from any interrupt context.

If **sema_init( )** or **sema_destroy( )** are used from interrupt context, lower-priority interrupts will not be serviced during the wait. This means that if the thread that will eventually perform the **sema_v( )** becomes blocked on anything that requires the lower-priority interrupt, the system will hang.

For example, the thread that will perform the **sema_v( )** may need to first allocate memory. This memory allocation may require waiting for paging I/O to complete, which may require a lower-priority disk or network interrupt to be serviced. In general, situations like this are hard to predict, so it is advisable to avoid waiting on semaphores or condition variables in an interrupt context.

SEE ALSO

kill(2), **condvar**(9F), **mutex**(9F)

Writing Device Drivers