3.2.15 AS_SEND

NAME

as_send -- "Send Signal to a Task"

SYNOPSIS

uint as_send (tid, signal)

```
uint tid;      /* task id as returned by t_create or t_ident */
uint signal;   /* signal set */
```

DESCRIPTION

The *as_send* directive sends signals to a task. The *signal* field describes the set of signals it wishes to send. Thirty-two signals are available. Sixteen are available as *system* signals and sixteen are available as *user* signals.

The signal set must be sent to tasks which have specified an asr using the *as_catch* directive. If the task identified by the *tid* does not have a valid asr, the caller returns with the invalid asr error.

When a signal is sent to a task with a valid and enabled asr, the task will be dispatched to the asr address when it becomes the running task. Signals sent to a blocked task are latched until the task becomes the running task. Duplicate signals are not queued.

The task identified by the *tid* may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the task was created with the GLOBAL flags value set (see *t_create*).

RETURN VALUE

If the *as_send* directive successfully sent the signal, then 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Invalid *tid*.

Invalid asr.

ISR cannot reference remote node.

NOTES

Can be called from within an ISR, except when the task was not created from the local node.

3.2.15 AS_SEND

NAME

as_send -- "Send Signal to a Task"

SYNOPSIS

uint as_send (tid, signal)

```
uint tid;      /* task id as returned by t_create or t_ident */
uint signal;   /* signal set */
```

DESCRIPTION

The *as_send* directive sends signals to a task. The *signal* field describes the set of signals it wishes to send. Thirty-two signals are available. Sixteen are available as *system* signals and sixteen are available as *user* signals.

The signal set must be sent to tasks which have specified an asr using the *as_catch* directive. If the task identified by the *tid* does not have a valid asr, the caller returns with the invalid asr error.

When a signal is sent to a task with a valid and enabled asr, the task will be dispatched to the asr address when it becomes the running task. Signals sent to a blocked task are latched until the task becomes the running task. Duplicate signals are not queued.

The task identified by the *tid* may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the task was created with the GLOBAL flags value set (see *t_create*).

RETURN VALUE

If the *as_send* directive successfully sent the signal, then 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Invalid *tid*.

Invalid asr.

ISR cannot reference remote node.

NOTES

Can be called from within an ISR, except when the task was not created from the local node.

3.2.16 AS_RETURN

NAME

as_return -- "Return from Signal Routine"

SYNOPSIS

void as_return ( )

DESCRIPTION

The *as_return* must be used by tasks to return from an asynchronous signal routine (asr).

RETURN VALUE

None.

ERROR CONDITIONS

Not in asr.

NOTES

This call is only used to return from an asr. Refer to the *as_catch* and *as_send* directives.

3.3 Semaphore Management

The semaphore manager provides a set of directives to use in arbitrating access to a shared resource (many-to-one). The semaphores primitives provided can be used to fulfill different sets of requirements:

1. To control access to a single resource that is either available or not, the user can create a semaphore with an initial value of 1.

2. To control access to a pool of "n" resources where at any moment "m" of those resources are available (0 <= m <= n) and "n-m" are not, the user can create a semaphore with an initial value of "n".

Arbitrating access to shared resources requires signaling that a predefined event has occurred. Sophisticated synchronisation also requires a counter to record the number of events sent but not yet received, and a list of tasks awaiting receipt of the event.

The semaphore data structure fulfills all the previous requirements. A semaphore possesses a name to distinguish it from the other semaphores within the system, a semaphore id to enable quick access to the semaphore, the requisite semaphore count variable to count the events, and a list of waiting tasks. In addition to the semaphore count variable, the semaphore contains an initial count, used as an initial assignment value for the semaphore count.

The synchronisation rules for semaphores ar :

1. The semaphore count is decremented by 1, when a task does a *sm_p* operation. The task continues execution if the count is then greater than or equal to zero. If the count is less than zero, the task is put on a waiting list for the semaphore.

2. The semaphore count is incremented by one when a task does a *sm_v* operation. If the count is less than or equal to zero, the first task in the semaphore waiting list is placed in the ready state.

The directives provided by the semaphore manager are:

Directive	Function
sm_create	Get a semaphore
sm_ident	Obtain the id of a Semaphore
sm_delete	Delete a semaphore
sm_p	Access semaphore
sm_v	Release semaphore

3.3.1 SM_CREATE

NAME

sm_create – "Create a Semaphore"

SYNOPSIS

```
#include <semaphore.h>
uint sm_create ( name, count, flags, &smid )

        uint name;    /* semaphore name */
        uint count;   /* initial count */
        uint flags;   /* semaphore flags */
        uint smid;    /* semaphore id - returned by this call */
```

The flags field values are:

PRIOR	set	to process wait list by priority
	clear	to process wait list by FIFO
GLOBAL	set	to indicate the semaphore is a multiprocessor global resource.
	clear	to indicate the semaphore is local.

DESCRIPTION

The *sm_create* directive creates a semaphore and assigns it an initial count equal to the value in the *count* field. The semaphore id is returned in *smid*. The *smid* must be used in subsequent *sm_p*, *sm_v*, and *sm_delete* calls.

By setting the PRIOR value in the flags field, tasks waiting on a semaphore will be processed in task priority order. Otherwise the tasks will be processed in first in, first out (FIFO) order.

By setting the GLOBAL value in the flags field, the *smid* will be sent to all processors in the system, to be entered into a global resource table. The system is defined as the collection of interconnected processors. The semaphore is always created on the local node.

The maximum number of semaphores that can be in existence at one time is a configuration parameter.

RETURN VALUE

If *sm_create* successfully created the semaphore, the *smid* is filled in, and 0 is returned.

If the semaphore was not successfully created, an error code is returned.

ERROR CONDITIONS

Too many semaphores.