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 describe: 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 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 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.

II 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 sernaphore 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 synchronization 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 synchronization rules for semaphore are:

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.

NOTES

Not callable from ISR.

Will not cause a preempt.

3.3.2 SM\_IDENT

NAME

sm\_ident – “Obtain the id of a Semaphore”

SYNOPSIS

#incIude <semaphore.h >

uint sm\_indent (name, node, &smid)

 uint name; /\* semaphore name \*/

 uint node; /\* node identifier \*/

 /\* 0 indicates any node \*/

 uint smid; /\* semaphore id – returned by this call \*/

DESCRIPTION

The sm\_ident directive allows a task to identify a previously created semaphore by name and receive the smid to use in sm\_p, sm\_v and sm\_delete directives for this semaphore.

If the semaphore name is not unique, the smid returned may not correspond to the semaphore named in this call.

The semaphore may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the semaphore was created with the GLOBAL flags value set (see sm\_create). If the semaphore name is not unique within the multiprocessor configuration, a non-zero node identifier must be specified in the node field.

RETURN VALUE

If sm\_ident succeeds, the smid will be filled in, and 0 is returned.

If sm\_ident does not succeed, an error code is returned.

ERROR CONDITIONS

Named semaphore does not exist.

Invalid node identifier.

NOTES

Can be called from within an ISR.

3.3.3 SM\_DELETE

NAME

sm\_delete - "Delete Semlphore"

SYNOPSIS

#include <semaphore.h>

uint sm\_delete (smid)

uint smid; /\* semaphore id as returned by sm\_create or sm\_ident \*/

DESCRIPTION

The semaphore identified by the smid is deleted from the system.

If tasks are waiting for the semaphore when the semaphore is deleted, each is made ready and given a return code indicating a deleted semaphore.

The semaphore must exist on the local processor. If the semaphore was created with the GLOBAL flags value set in a multiprocessor configuration, a notification will be sent to all processors in the system, so the smid can be deleted from the global resource table.

The requester does not have to be the creator of the semaphore. Any task knowing the smid can delete it.

RETURN VALUE

If sm\_delete successfully deleted the semaphore, 0 is returned.

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

ERROR CONDITIONS

Invalid smid.

Semaphore not created from local node.

NOTES

Not callable from ISR.

May cause a preempt if a task waiting for the semaphore has a higher priority than the running task, and the preempt mode is in effect. A preempt will not occur if all tasks waiting for the semaphore exist on a remote processor in a multiprocessor configuration.

3.3.4 SM\_P

NAME

sm\_p – “Access Semaphore”

Synopsis

#include <semaphore.h>

uint sm\_p (smid, flags, timeout )

uint smid; /\* semaphore id as returned by sm\_create or sm\_ident \*/

uint flags; /\* wait option \*/

 uint timeout; /\* number of ticks to wait \*/

 /\* 0 indicates wait forever \*/

The flags field values are:

 NOWAIT set return immediately with error if

 semaphore count is negative

clear wait for resource

DESCRIPTION

If the NOWAIT flags value is clear, the current semaphore count of the semaphore identified; by the smid is decremented by one. If the count is zero or positive, the requesting task continues execution, returning without error. If the count is negative, the requesting task must wait for access to the resource, and is put on a waiting list.

If the NOWAIT flags value is set, and the count is negative, an error is returned. If the count is zero or positive, zero is returned.

The semaphore identified by the smid may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the semaphore was created with the GLOBAL flags value set (see sm\_create).

When sm\_p is called from an ISR, the no-wait option is forced by the executive.

RETURN VALUE

If sm\_p succeeded, then 0 is returned.

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

ERROR CONDITIONS

Invalid smid.

Timeout ( if wait and timeout is selected ).

The semaphore count is negative ( if no wait is selected ).

Semaphore deleted.

ISR cannot reference remote node.

NOTES

Can be called from within an ISR, except when the semaphore was not created on the local node. The no-wait option is forced by the executive.

The running tank will be blocked if the count in negative.