6.3. SEM_IDENT

Obtain the identifier of a semaphore on a given node with a given name.

Synopsis

```
sem_ident( name, nid, sid )
```

Input Parameters

name	: string	user defined semaphore name
nid	: node_id	node identifier

Output Parameters

sid	: sema_id	kernel defined semaphore identifier

Literal Values

nid	= LOCAL_NODE	the node containing the calling task
	= OTHER_NODES	all nodes in the system except the local node.

Completion Status

OK	sem_ident operation successful
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_NODE	node does not exist
NAME_NOT_FOUND	name does not exist on node
NODE_NOT_REACHABLE	node on which semaphore resides is not reachable

Description

This operation searches the kernel data structure in the node(s) specified for a semaphore with the given name, and returns its identifier if found. If OTHER_NODES is specified, the node search order is implementation dependent. If there is more than one semaphore with the same name in the node(s) specified, then the sid of the first one found is returned.

6.4. SEM_P

Perform P operation (take) on a semaphore.

Synopsis

```
   sem_p( sid, options, time_out )
```

Input Parameters

sid	: sema_id	kernel defined semaphore identifier
options	: bit_field	semaphore wait options
time_out	: integer	ticks to wait before timing out

Output Parameters

<none>

Literal Values

options	+ NOWAIT	do not wait - return immediately if semaphore not available
time_out	= FOREVER	wait forever - do not time out

Completion Status

OK	sem_p operation successful
ILLEGAL_USE	operation not callable from ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	semaphore does not exist
OBJECT_DELETED	semaphore specified has been deleted
TIME_OUT	sem_p operation timed out
SEMAPHORE_DELETED	semaphore deleted while blocked in sem_p operation
SEMAPHORE_NOT_AVAILABLE	semaphore unavailable with NOWAIT option
NODE_NOT_REACHABLE	node on which semaphore resides is not reachable

Description

This operation performs a claim from the given semaphore. It first checks if the NOWAIT option has been specified and the counter is zero or less, in which case the SEMAPHORE_NOT_AVAILABLE completion status is returned. Otherwise, the counter is decreased. If the counter is now zero or more, then the claim is successful, otherwise the calling task is put on the semaphore queue.

If the semaphore is deleted while the task is waiting on its queue, then the task is unblocked and this operation returns the SEMAPHORE_DELETED completion status. Otherwise the task is blocked either until the timeout expires, in which case the TIME_OUT completion status is returned, or until the task reaches the head of the queue and another task performs a sem_v operation on this semaphore.

6.5. SEM_V

Perform a V operation (give) on a semaphore.

Synopsis

```
sem_v( sid )
```

Input Parameters

sid	: sema_id	kernel defined semaphore identifier

Output Parameters

<none>

Completion Status

OK	sem_v operation successful
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	semaphore does not exist
OBJECT_DELETED	semaphore specified has been deleted
SEM_OVERFLOW	the counter of semaphore overflows
NODE_NOT_REACHABLE	node on which semaphore resides is not reachable

Description

This operation increments the semaphore count by one. If the resulting semaphore count is less than or equal to zero then the first task in the semaphore queue is unblocked, and returned the successful completion status.

6.6. SEM_INFO

Obtain information on a semaphore.

Synopsis

```
   sem_info( sid, options, count, tasks_waiting )
```

Input Parameters

sid	: sem-id	kernel defined semaphore identifier

Output Parameters

options	: bit_field	semaphore create options
count	: integer	semaphore count at time of call
tasks_waiting	: integer	number of tasks waiting in the semaphore queue

Completion Status

OK	sem_info operation successful
ILLEGAL_USE	operation not callable from ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	semaphore does not exist
OBJECT_DELETED	semaphore specified has been deleted
NODE_NOT_REACHABLE	node on which semaphore resides is not reachable

Description

This operation provides information on the specified semaphore. It returns its create options, the value of it's counter, and the number of tasks waiting on the semaphore queue. The latter two values should be used with care as they are just a snap-shot of the semaphores's state at the time of executing the operation.

7. QUEUES

Queues permit the passing of messages amongst tasks. Queues contain a variable number of messages, all of which have the same user task defined length. The queues normally behave first in first out, with messages sent to a queue being appended at the tail, and messages received from a queue being taken from the head. Urgent messages can be inserted at the head of the queue, i.e. they are prepended. Several urgent messages prepended without an intervening receive will be received last in first out.

Queue Behavior

The following should not be understood as a recipe for implementations.

When a queue contains no messages, a task which receives from it is blocked (unless it specified the NOWAIT option) and is put on the queue's wait queue. This queue of waiting tasks is ordered either by task priority or as first in first out.

A task may broadcast a message to all tasks on a wait queue, which unblocks all of them and returns them all the same message. This latter operation is atomic with respect to any other operation on this queue.

When a message is sent to a queue, the message data is immediately copied by the kernel. If no task is waiting for a message from the queue when one is sent, then the kernel copies the message into a buffer. If a task is waiting when one is sent, then the message may be copied into a buffer or it may be delivered directly to the waiting task. Whether a buffer is used in this case is implementation dependent.

All messages in a queue may be flushed with a single operation that is atomic with respect to any other operation on this queue.

Observation:

It can be seen that there is more than one way to use a queue. At one extreme, many tasks feed messages onto a queue and a single task receives them, creating a many to one data flow. At the other extreme, many tasks wait for a message and one task broadcasts a message synchronously to all of them, creating a one to many data flow.

Queue Options

A queue's options are set by the creating task. They define various aspects of the behavior of the kernel with respect to queues. ORKID defines the following option symbols, which may be combined unless otherwise stated. An implementation may define additional options.