6.6. POOL_INFO

Obtain information on a pool.

Synopsis

```
pool_info( pid, buffers, free_buffers, buff_size, options )
```

Input Parameters

pid	: pool-id	kernel defined pool identifier

Output Parameters

buffers	: integer	number of buffers in the pool
free_buffers	: integer	number of free buffers in the pool
buff_size	: integer	pool buffer size in bytes
options	: bit_field	pool create options

Completion Status

OK	pool_info successful
ILLEGAL_USE	pool_info not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_ID	pool does not exist
OBJECT_DELETED	originally existing pool has been deleted before operation
NODE_NOT_REACHABLE	node on which the pool resides is not reachable

Description

This operation provides information on the specified pool. It returns its overall number of buffers, the number of free buffers in the pool, its buffer size in bytes and options. The number of free buffers in the pool should be used with care as it is just a snap-shot of the pools's usage at the time of executing the operation.

7. SEMAPHORES

The semaphores defined in **ORKID** are standard Dijkstra counting semaphores. Semaphores provide for the fundamental need of synchronization in multi-tasking systems, i.e. mutual exclusion, resource management and sequencing.

Semaphore Behavior

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

During a sem_claim operation, the semaphore count is decremented by one. If the resulting semaphore count is greater than or equal to zero, then the calling task continues to execute. If the count is less than zero, the task blocks from processor usage and is put on a waiting queue for the semaphore. During a sem_release operation, the semaphore count is incremented by one. If the resulting semaphore count is less than or equal to zero, then the first task in the waiting queue for this semaphore is unblocked and is made eligible for processor usage.

Semaphore Usage

Mutual exclusion is achieved by creating a counting semaphore with an initial count of one. A resource is guarded with this semaphore by requiring all operations on the resource to be proceeded by a sem_claim

operation. Thus, if one task has claimed a resource, all other tasks requiring the resource will be blocked until the task releases the resource with a sem_release operation.

In situations where multiple copies of a resource exist, the semaphore may be created with an initial count equal to a number of copies. A resource is claimed with the sem_claim operation. When all available copies of the resource have been claimed, a task requiring the resource will be blocked until return of one of the claimed copies is announced by a sem_release operation.

Sequencing is achieved by creating a semaphore with an initial count of zero. A task may pend the arrival of another task by performing a sem_claim operation when it reaches a synchronization point. The other task performs a sem_release operation when it reaches its synchronization point, unblocking the pending task.

Semaphore Options

ORKID defines the following option symbols, which may be combined.

+ GLOBAL Semaphores created with the GLOBAL option set are visible and accessible from any node in the system.

+ FIFO Semaphores with the FIFO option set enter additional tasks at the end of their waiting queue. Without this option, the tasks are enqueued in order of task priority. **ORKID** does not require reordering of semaphore waiting queues when a waiting task has his priority changed.

7.1. SEM_CREATE

Create a semaphore.

Synopsis

```
sem_create( name, init_count, options, sid )
```

Input Parameters

name	: string	user defined semaphore name
init_count	: integer	initial semaphore count
options	: bit_field	semaphore create options

Output Parameters

sid	: sem_id	kernel defined semaphore identifier

Literal Values

options	+ GLOBAL	the new semaphore will be visible throughout the system
	+ FIFO	tasks will be queued in first in first out order

Completion Status

OK	sem_create successful
ILLEGAL_USE	sem_create not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_COUNT	initial count is negative
INVALID_OPTIONS	invalid options value
TOO_MANY_OBJECTS	too many semaphores on the node or in the system

Description

This operation creates a new semaphore in the kernel data structure, and returns its identifier. The semaphore is created with its count at the value given by the init_count parameter. The task queue, initially empty, will be ordered by task priority, unless the FIFO option is set, in which case it will be first in first out.

7.2. SEM_DELETE

Delete a semaphore.

Synopsis

```
sem_delete( sid )
```

Input Parameters

sid	: sem_id	kernel defined semaphore identifier

Output Parameters

<none>

Completion Status

OK	sem_delete successful
ILLEGAL_USE	sem_delete not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_ID	semaphore does not exist
OBJECT_DELETED	originally existing semaphore has been deleted before operation
OBJECT_NOT_LOCAL	sem_delete not allowed on non-local semaphore

Description

The sem_delete operation deletes a semaphore from the kernel data structure. The semaphore is deleted immediately, even though there are tasks waiting in its queue. These latter are all unblocked and are returned the SEMAPHORE_DELETED completion status.

7.3. 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	: sem_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
	= ALL_NODES	all nodes in the system

Completion Status

OK	sem_ident successful
ILLEGAL_USE	sem_ident not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_ID	node does not exist
NAME_NOT_FOUND	semaphore does not exist on node
NODE_NOT_REACHABLE	node 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 or ALL_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.