6.1. POOL\_CREATE

Create a pool.

Synopsis

pool\_create( name, addr, length, buff\_size, options, pid )

Input Parameters

name : string user defined pool name

addr : address start address of pool

length : integer length of pool in bytes

buff\_size : integer pool buffer size in bytes

options : bit\_field pool create options

Output Parameters

pid : pool\_id kernel defined pool identifier

Literal Values

options + GLOBAL pool is global within the shared memory subsystem

+ FORCED\_DELETE deletion will go ahead even if there are unrealeased buffers

Completion Status

OK pool\_create successful

ILLEGAL\_USE pool\_create not callable from ISR

INVALID\_PARAMETER a parameter refers to an invalid address

INVALID\_BUFF\_SIZE buff\_size not supported

INVALID\_OPTIONS invalid options value

TOO\_MANY\_OBJECTS too many pools on the node or in the system

POOL\_0VERLAP area given overlaps an existing pool

Description

This operation declares an area of memory to be organized as a pool by the kernel. The process of formatting the memory to operate as a pool may require a memory overhead which may be taken from the new pool. It can never be assumed that all of the memory in the pool will be available for allocation. The overhead percentage will be implementation dependent.

The FORCED\_DELETE option governs the deletion possibility of the pool (see 6.2 pool\_delete)

6.2. POOL\_DELETE

Delete a pool.

Synopsis

pool\_delete( pid )

Input Parameters

pid : pool\_id kernel defined pool identifier

Output Parameters

<none>

Completion Status

OK pool\_delete successful

ILLEGAL\_USE pool\_delete 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

POOL\_IN\_USE buffers from this pool are still allocated

OBJECT\_NOT\_LOCAL pool\_delete not allowed on non—local pools

Description

Unless the FORCED\_DELETE option was specified at creation, this operation first checks whether the pool has any buffers which have not been returned. If this is the case, then the POOL\_IN\_USE completion status is returned. If not, and in any case if FORCED\_DELETE was specified, then the pool is deleted from the kernel data structure.

6.3. POOL\_IDENT

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

Synopsis

pool\_ident( name, nid, pid)

Input Parameters

name : string user defined pool name

nid : node\_id node identifier

Output Parameters

pid : pool\_id kernel defined pool 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 pool\_ident successful

ILLEGAL\_USE pool\_ident not callable from ISR

INVALID\_PARAMETER a parameter refers to an invalid address

INVALID\_ID node does not exist

NAME\_NOT\_FOUND pool 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 pool 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 pool with the same name, then the pid of the first one found is passed back.

Observation:

*This operation may return the pid of a GLOBAL pool that is not in the same shared memory subsystem as the node containing the calling task.*

6.3. POOL\_GET\_BUFF

Get a buffer from a pool.

Synopsis

pool\_get\_buff( pid, buff\_addr )

Input Parameters

pid : pool\_id kernel defined pool identifier

Output Parameters

buff\_addr : address address of obtained buffer

Completion Status

OK pool\_get\_buff successful

ILLEGAL\_USE pool\_get\_buff not callable from ISR

INVALID\_PARAMETER a parameter refers to an invalid address

INVALID\_ID pool does not exist

OBJECT\_DELETED originally existing task has been deleted before operation

NO\_MORE\_MEMORY no more buffers available in pool

POOL\_NOT\_SHARED pool not in shared memory subsystem

NODE\_NOT\_REACHABLE node on which pool resides is not reachable

Description

The pool\_get\_buff requests for a single buffer from the pool's free memory. If the kernel cannot immediately fulfil the request, it returns the completion status NO\_MORE\_MEMORY, otherwise the address of the allocated buffer is returned. The exact allocation algorithm is implementation dependent.

6.5. POOL\_RET\_BUFF

Return a buffer to its pool.

Synopsis

pool\_ret\_buff( pid, buff\_addr)

Input Parameters

pid : pool\_id kernel defined pool identifier

buff\_addr : address address of buffer to be returned

Output Parameters

<none>

Completion Status

OK pool\_ret\_buff successful

ILLEGAL\_USE pool\_ret\_buff 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

POOL\_NOT\_SHARED pool not in shared memory sybsystem

INVALID\_BUFF no buffer allocated from pool at buff\_addr

NODE\_NOT\_REACHABLE node on which pool resides is not reachable

Description

This operation returns the given buffer to the given pool's free space. The kernel checks that the buffer was previously allocated from the pool and returns INVALID\_BUFF if it wasn't.

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

0BJECT\_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.