5.5. PARTITION_RET_BLK

Return a block to its partition.

Synopsis

```
   partition_ret_blk( pid, blk_addr )
```

Input Parameters

pid	: part_id	kernel defined partition identifier
blk_addr	: address	address of block to be returned

Output Parameters

<none>

Completion Status

OK	partition_ret_blk operation successful
ILLEGAL_USE	operation not callable from ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	partition does not exist
OBJECT_DELETED	partition specified has been deleted
INVALID_BLOCK	no block allocated from partition at blk_addr
NODE_NOT_REACHABLE	node on which task resides is not reachable

Description

This operation returns the given block to the given partition's free block pool. The kernel checks that the block was previously allocated from the partition and returns INVALID_BLOCK if it wasn't.

5.6. PARTITION_INFO

Obtain information on a partition.

Synopsis

```
   partition_info( pid, blocks, free_blocks, block_size  )
```

Input Parameters

pid	: partition-id	kernel defined region id

Output Parameters

blocks	: integer	number of blocks in the partition
free_blocks	: integer	number of free blocks in the partition
block_size	: integer	partition block size in bytes

Completion Status

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

Description

This operation provides information on the specified partition. It returns its overall number of blocks, the number of free blocks in the partition, and the block size. The number of free blocks in the partition should be used with care as it is just a snap-shot of the partitions's usage at the time of executing the operation.

6. 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.

The behavior of counting semaphores can be described as follows:

During a sem_p operation, the semaphore count is decremented by one. If the resulting semaphore count is greater than or equal to zero, than the calling task continues to execute. If the count is less than zero, the task blocks from CPU usage and is put on a waiting list for the semaphore.

During a sem_v 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 list for this semaphore is unblocked and is made eligible for CPU 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_p 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_v operation.

In situations where multiple instantiations of a resource exist, the semaphore may be created with an initial count equal to a number of instantiations. A resource is claimed from the pool with the sem_p operation. When all available copies of the resource have been claimed, a task requiring the resource will be blocked until one of the claimed resources is returned to the pool by a sem_v 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_p operation when it reaches a synchronization point. The other tasks performs a sem_v operation when it reaches its synchronization point, unblocking the pended 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 created with the FIFO option set enqueue blocked tasks in order of arrival of the sem_p

operations. Without this option, the tasks are enqueued in order of task priority.

6.1. 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	:	sema_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 operation successful
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_COUNT	init count is negative
INVALID_OPTIONS	invalid options value
TOO_MANY_SEMAPHORES	too many semaphores on node

Description

This operation creates a new semaphore in the kernel data structure, and returns its identifier. The semaphore is created with its counter at the value given by the 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.

6.2. SEM_DELETE

Delete a semaphore.

Synopsis

```
   sem_delete( sid )
```

Input Parameters

sid	: sema_id	kernel defined semaphore identifier

Output Parameters

<none>

Completion Status

OK	sem_delete 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

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.

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.

- GLOBAL Queues created with the GLOBAL option set are visible and accessible from any node in the system. When a message is sent to a queue in another node, the message is physically copied to that other node. In non-shared memory systems, it is not guaranteed that a message has arrived in the destination node before the operation returns a successful completion status.

- FIFO With this option set, the tasks waiting for messages from the queue will be queued first in first out. The tasks are by default queued in order of task priority.

7.1. QUEUE_CREATE

Create a message queue.

Synopsis

```
queue_create( name, max_buff, length, options, qid )
```

Input Parameters

name	: string	user defined queue name
max_buff	: integer	maximum number of buffers allowed in queue
length	: integer	length of message buffers in bytes
options	: bit_field	queue create options

Output Parameters

qid	: queue_id	kernel defined queue identifier

Literal Values

options	+ GLOBAL	the new queue will be visible throughout the system
	+ FIFO	tasks waiting on a message will be queued first in first out

Completion Status

OK	queue_create operation successful
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_LENGTH	buffer length not supported
INVALID_OPTIONS	invalid options value
TOO_MANY_QUEUES	too many queues on node
NO_MORE_MEMORY	not enough memory to allocate message buffer(s)

Description

This operation creates a new queue in the kernel data structure. The given number of buffers of the given length are allocated by the kernel. If the kernel cannot find sufficient memory it returns the NO_MORE_MEMORY completion status.

The maximum possible length of messages is implementation dependent, but an ORKID compliant kernel is required to support message lengths of up to 32 bytes.

7.2. QUEUE_DELETE

Delete an existing queue.

Synopsis

```
queue_delete( qid )
```

Input Parameters

qid	: queue_id	kernel defined queue identifier

Output Parameters

<none>

Completion Status

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

Description

This option deletes the given queue from the kernel data structure. If any tasks were waiting for a message from the queue, they are unblocked and returned the QUEUE_DELETED completion status. If there were any messages in the queue, they are lost and the buffers deallocated.

7.3. QUEUE_IDENT

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

Synopsis

```
   queue_ident( name, nid, qid )
```

Input Parameters

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

Output Parameters

qid	: queue_id	kernel defined queue 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	queue_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 queue 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 queue with the same name in the node(s) specified, then the qid of the first one found is returned.

7.4. QUEUE_SEND

Send a message to a given queue.

Synopsis

```
queue_send( qid, message, length )
```

Input Parameters

qid	: queue_id	kernel defined queue identifier
message	: address	message starting address
length	: integer	length of message in bytes

Output Parameters

<none>

Completion Status

OK	queue_send operation successful
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	queue does not exist
OBJECT_DELETED	queue specified has been deleted
INVALID_LENGTH	message length greater than queue's buffer length
QUEUE_FULL	no more buffers available
NODE_NOT_REACHABLE	node on which semaphore resides is not reachable

Description

This operations sends a message to a queue. If the queue's wait queue contains a number of tasks waiting on messages, then the message is delivered to the task at the head of the wait queue. This task is then removed from the wait queue, unblocked and will be returned a successful completion status along with the message. Otherwise the message is put on the queue.

If the maximum queue length has been reached, then the QUEUE_FULL completion status is returned.

7.5. QUEUE_URGENT

Send a message to head of queue.

Synopsis

```
queue_urgent( qid, message, length )
```

Input Parameters

qid	: queue_id	kernel defined queue identifier
message	: address	message starting address
length	: integer	message length in bytes

Output Parameters

<none>

Completion Status

OK	queue_urgent operation successful
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	queue does not exist
OBJECT_DELETED	queue specified has been deleted
INVALID_LENGTH	message length greater than queue's buffer length
QUEUE_FULL	no more buffers available
NODE_NOT_REACHABLE	node on which semaphore resides is not reachable

Description

This operation sends a priority message to a queue.

If the queue's wait queue contains a number of tasks waiting on messages, then the action is exactly the same as for queue send. The message is delivered to the task at the head of the wait queue. This task is then removed from the wait queue, unblocked and will be returned a successful completion status along with the message.

Otherwise the message is inserted at the head of the message queue. If there is no memory available for the buffer, then the NO_MORE_MEMORY completion status is returned.

7.6. QUEUE_BROADCAST

Broadcast message to all tasks blocked on a queue.

Synopsis

```
queue_broadcast( qid, message, length, count )
```

Input Parameters

qid	: queue_id	kernel defined queue identifier
message	: address	message starting address
length	: integer	message length in bytes

Output Parameters

count	: integer	number of unblocked tasks

Completion Status

OK	queue_broadcast operation successful
ILLEGAL_USE	operation not callable from ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	queue does not exist
OBJECT_DELETED	queue specified has been deleted
INVALID_LENGTH	message length greater than queue's buffer length
NODE_NOT_REACHABLE	node on which semaphore resides is not reachable

Description

This operation sends a message to all tasks waiting on the queue. If the wait queue is empty, then no messages are sent, no tasks are unblocked and the count returned will be zero. If the wait queue contains a number of tasks waiting on messages, then the message is delivered to each task in the wait queue. All tasks are then removed from the wait queue, unblocked and returned a successful completion status. The number of tasks unblocked is returned in the count parameter.

This operations is atomic with respect to other operations on the queue.

7.7. QUEUE_RECEIVE

Receive a message from a queue.

Synopsis

```
queue_receive( qid, message, options, time_out )
```

Input Parameters

qid	: queue_id	kernel defined queue identifier
message	: address	address to put message
options	: bit_field	queue receive options
time_out	: integer	max number of ticks to wait

Output Parameters

<none>

Literal Values

options	+ NOWAIT	do not wait - return immediately if no message in queue
time_out	= FOREVER	wait forever - do not time out

Completion Status

OK	queue_receive operation successful
ILLEGAL_USE	operation not callable from ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	queue does not exist
OBJECT_DELETED	queue specified has been deleted
INVALID_ADDRESS	message refers to an illegal address
INVALID_OPTIONS	invalid options value
TIME_OUT	queue-receive operation timed out
QUEUE_DELETED	queue deleted while blocked in queue_receive operation
QUEUE_EMPTY	queue empty with NOWAIT option
NODE_NOT_REACHABLE	node on which semaphore resides is not reachable

Description

This operation receives a message from a given queue. If there are one or more messages on the queue, then the buffer at the head is removed from the queue, its message is copied into the given area, the buffer is deallocated, and a successful completion status returned.

If the queue is empty, and NOWAIT was not specified in the options, then the task is blocked and put on the queue's wait queue in order of task priority or first in first out. If NOWAIT was specified and the queue is empty, then the QUEUE_EMPTY completion status is returned. If the queue is deleted while the task is waiting on a message from it, then the QUEUE_DELETED completion status is returned. If the timeout expires, then the TIME_OUT completion status is returned. Otherwise, when the task reaches the head of the queue and a message is sent, or if a message is broadcast while the task is anywhere in the queue, then the task receives the message and is returned a successful completion status.