3.10. TASK_READ_NOTE_PAD

Read one of a task's note-pad locations.

Synopsis

```
task_read_note_pad( tid, loc_number, loc_value )
```

Input Parameters

tid	: task_id	kernel defined task id
loc_number	: lnum	note-pad location number

Output Parameters

loc_value	: integer	note-pad location value

Literal Values

tid	= SELF	The calling task reads its own notepad

Completion Status

OK	task_read_note_pad operation successful
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	task does not exist
INVALID_LOCATION	note-pad number does not exist
NODE_NOT_REACHABLE	node on which task resides is not reachable

Description

This operation returns the value contained in the specified notepad location of the task identified by tid. (see also 3. Task Notepads)

3.11. TASK_WRITE_NOTE_PAD

Write one of a task's note-pad locations.

Synopsis

```
   task_write_note_pad( tid, loc_number, loc_value )
```

Input Parameters

tid	: task_id	kernel defined task id
loc_number	: lnum	note-pad location number
loc_value	: integer	note-pad location value

Output Parameters

<none>

Literal Values

tid	= SELF	The calling task writes into its own notepad

Completion Status

OK	task_write_note_pad operation successful
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	task does not exist
OBJECT_DELETED	task specified has been deleted
INVALID_LOCATION	note-pad number does not exist
NODE_NOT_REACHABLE	node on which task resides is not reachable

Description

This operation writes the specified value into the specified notepad location of the task identified by tid. (see also 3. Task Notepads)

4. REGIONS

A region is an area of memory within a node which is organized by an **ORKID** compliant kernel into a pool of segments of varying size. The area of memory to become a region is declared to the kernel by a task when the region is created, and is thereafter managed by the kernel until it is explicitly deleted by a task.

Each region has a granularity, defined when the region is created. The actual size of segments allocated is always a multiple of the granularity, although the required segment size is given in bytes.

Once a region has been created, a task is free to claim variable sized segments from it and return them in any order. The kernel will do its best to satisfy all requests for segments, although fragmentation may cause a segment request to be unsuccessful, despite there being more than enough total memory remaining in the region. The memory management algorithms used are implementation dependent.

Regions, as opposed to partitions, tasks, etc., are only locally accessible. In other words, regions cannot be declared global and a task cannot access a region on another node. This does not stop a task from using the memory in a region on another node, for example in an area of memory shared between the nodes, but all claiming of segments must be done by a co-operating task in the appropriate node and the address passed back.

Observation:

Regions are intended to provide the first subdivisions of the physical memory available to a node. These subdivisions may reflect differing physical nature of the memory, giving for example a region of RAM, a region of ROM, a region of shared memory, etc.. Regions may also subdivide memory into areas for different uses, for example a region for kernel use and a region for user task use.

4.1. REGION_CREATE

Create a region.

Synopsis

```
region_create( name, addr, length, granularity, options, rid )
```

Input Parameters

name	: string	user defined region name
addr	: address	start address of the region
length	: integer	length of region in bytes
granularity	: integer	allocation granularity in bytes
options	: bit_field	region create options

Output Parameters

rid	: region_id	kernel defined region identifier

Completion Status

OK	region_create operation successful
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ADDRESS	area given not within actual memory present
INVALID_GRANULARITY	granularity not supported
INVALID_OPTIONS	invalid options value
TOO_MANY_REGIONS	too many regions on the node
REGION_OVERLAP	area given overlaps an existing region

Description

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

Observation:

*Currently **ORKID** defines no options, the parameter is there as a place holder for future extensions and implementations desiring to provide additional options.*

4.2. REGION_DELETE

Delete a region.

Synopsis

```
   region_delete( rid, options )
```

Input Parameters

rid	: region_id	kernel defined region identifier
options	: bit_field	region deletion options

Output Parameters

<none>

Literal Values

options	+ FORCED_DELETE	deletion will go ahead even though there are unreleased segments

Completion Status

OK	region_delete operation successful
ILLEGAL_USE	operation not callable from ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	region does not exist
OBJECT_DELETED	region specified has been deleted
INVALID_OPTIONS	invalid options value
REGION_IN_USE	segments from this region are still allocated

Description

Unless the FORCED_DELETE option was specified, this operation first checks whether the region has any segments which have not been returned. If this is the case, then the REGION_IN_USE completion status is returned. If not, and in any case if FORCED_DELETE was specified, then the region is deleted from the kernel data structure.