3. TASKS

Tasks are single threads of program execution. Within a node, a number of tasks may run concurrently, competing for CPU time and other resources. **ORKID** does not define the number of tasks allowed per node. Tasks are created and deleted dynamically by existing tasks.

Tasks are allocated CPU time by a part of the kernel called the scheduler. The exact behavior of the scheduler is implementation dependent, but it must have the minimum functionality described in the following paragraphs.

Throughout its existence, each task has a current priority, a current mode and a current state, all of which may change over time. A task may also have an exception service routine which has to be declared to it at runtime.

Task Exception Service Routine

A task may designate an Exception Service Routine (XSR) to handle exceptions which have been sent to that task. A task's XSR can be changed at will, but a task can have only one at any time. The purpose of an XSR is to deal with exceptions which have been sent to the task. It is recommended that exceptions be reserved for errors and other abnormal conditions which arise.

A task's XSR is activated asynchronously. This means that it is not called explicitly by the task code, but automatically by the scheduler whenever one or more exceptions are sent to the task. Thus an XSR may be entered at any time during task execution. (But see 'Task Modes' below.) A task's XSR runs at least at the same priority as the task; it only needs to be executed when the task normally would have been scheduled to the running state. Exceptions are latched on a single level. Multiple occurrences of the same exception during this time will be seen as a single exception by the XSR.

Task Priority

A task's priority determines its 'importance' in relation to the other tasks within the node. Priority is a numeric parameter and can take any value in the range 1 to HIGHP. Priority HIGHP is 'highest' or 'most important' and priority 1 is 'lowest' or 'least important'. There may be any number of tasks with the same priority.

Priorities are assigned to tasks by the tasks themselves, and affect the way in which task scheduling occurs. Although the exact scheduling algorithm is outside the scope of this standard, in general the higher the priority of a task, the more likely it is to receive CPU time.

Task Modes

A task's mode determines certain aspects of the behavior of the kernel in respect to the task. The mode is made up by the combination of a number of mode parameters, each of which determines a single aspect of kernel behavior.

3.1. TASK_CREATE

Create a task.

Synopsis

```
task_create( name, priority, stack_size, mode, options, tid )
```

Input parameters

name	: string	user defined task name
priority	: prio	initial task priority
stack_size	: integer	size in bytes of task's stack
mode	: bit_field	initial task mode
options	: bit_field	creation options

Output Parameters

tid	: task_id	kernel defined task identifier

Literal Values

mode	+ NOXSR	XSRs cannot be activated
	+ NOTERMINATION	task cannot be restarted or deleted
	+ NOPREEMPT	task cannot be preempted
	+ NOINTERRUPT	interrupt handling routine cannot be activated
options	+ GLOBAL	New task will be visible throughout the system.

Completion Status

OK	task_create operation successful
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_PRIORITY	invalid priority value
INVALID_MODE	invalid mode value
INVALID_OPTIONS	invalid options value
TOO_MANY_TASKS	too many tasks on the node
NO_MORE_MEMORY	not enough memory to allocate task data structure or task stack

Description

The task_create operation creates a new task in the kernel data structure. Tasks are always created in the node in which the call to task_create was made. The new task does not start executing code - this is achieved with a call to the task_start operation.
The tid returned by the kernel is used in all subsequent **ORKID** operations (except task_ident) to identify the newly created task. If GLOBAL is specified in the options parameter, then the tid can be used anywhere in the system to identify the task, otherwise it can be used only in the node in which the task was created.

3.2. TASK_DELETE

Delete a task.

Synopsis

task_delete(tid)

Input Parameters

tid	: task_id	kernel defined task identifier

Output Parameters

<none>

Literal Values

tid	= SELF	The calling task requests its own deletion.

Completion Status

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

Description

This operation stops the task identified by the tid parameter and deletes it from its node's kernel data structure. If the task's active mode has the parameters NOTERMINATION set, then the task will not be deleted and the completion status OBJECT_PROTECTED will be returned.

Observation:

The task_delete operation performs no 'clean-up' of the resources allocated to the task. It is therefore the responsibility of the calling task to ensure that all segments, blocks, etc., allocated to the task to be deleted have been returned.

For situations where one task must delete another, clean-up will usually require co-operation between the tasks, typically using exceptions, or task_restart.

3.3. TASK_IDENT

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

Synopsis

```
task_ident( name, nid, tid )
```

Input Parameters

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

Output Parameters

tid	: task_id	kernel defined task identifier

Literal Values

nid	= LOCAL_NODE	The node containing the calling task
	= OTHER_NODES	all nodes in the system except the local node
name	= WHO_AM_I	Returns tid of calling task

Completion Status

OK	task_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 task resides is not reachable

Description

This operation searches the kernel data structure in the node(s) specified by nid for a task with the given name. If OTHER_NODES is specified, the node search order is implementation dependent. If there is more than one task with the same name in the node(s) specified, then the tid of the first one found is returned.

3.4. TASK_START

Start a task.

Synopsis

```
task_start( tid, start_addr, arguments )
```

Input Parameters

tid	: task_id	kernel defined task identifier
start_addr	: *	task start address
arguments	: *	arguments passed to task

Output Parameters

<none>

Completion Status

OK	task_start operation successful
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	task does not exist
OBJECT_DELETED	task specified has been deleted
INVALID_ADDRESS	invalid start address
INVALID_ARGUMENTS	invalid number or type or size of arguments
TASK_ALREADY_STARTED	task has been started already
OBJECT_PROTECTED	task has NOTERMINATION parameter set
NODE_NOT_REACHABLE	node on which task resides is not reachable

Description

The task_start operation starts a task at the given address. The task must have been previously created with the task_create operation. The task is started with the priority and mode specified when the task was created.

* The specification of start address and the number and type of arguments are language binding dependent. For a high level language, the start address will likely be the name of a procedure and the arguments would be passed to the procedure as parameters.