3.2. NODE_FAIL

Indicates fatal node failure to the system.

Synopsis

```
node_fail( nid, code, options )
```

Input Parameters

nid	: node_id	system defined node identifier
code	: integer	type of error detected
options	: bit_field	failure options

Output Parameters

<none>

Literal Values

options	+ TOTAL	all nodes should be stopped

Completion Status

OK	node_fail successful
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_ID	node does not exist
OBJECT_NOT_LOCAL	node_fail on remote node not allowed from ISR
NODE_NOT_REACHABLE	node is not reachable

Description

This operation indicates a fatal failure of the type given by code in the node identified by nid to the system. If the TOTAL option is set all nodes of the system should be stopped, otherwise only the node identified by nid is stopped. The operation does not return if, as a result of the operation, the local node is stopped.

Observation:

The value in code can be transferred to a certain memory location or even displayed by hardware in the failing node to ease post mortem analysis of the failure.

3.3 NODE_INFO

Obtain information on a node.

Synopsis

```
node_info( nid, ticks_per_sec )
```

Input Parameters

nid	: node_id	system defined node identifier

Output Parameters

ticks_per_sec	: integer	number of ticks per second for node clock

Completion Status

OK	node_info successful
ILLEGAL_USE	node_info not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_ID	node does not exist
NODE_NOT_REACHABLE	node is not reachable

Description

This operation obtains the number of ticks per second for the clock on the node identified by nid.

Observation:

For efficiency all delay times are specified in ticks. The value of ticks_per_sec allows tasks to convert between seconds and ticks.

4. 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 or in a system. 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 lifetime, each task has a current priority and a current mode, 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 Exception Service Routine (XSR) to handle exceptions which have been raised for that task. A task can have one XSR defined for every bit in the exception bit-field. XSRs can be redefined dynamically. The purpose of XSRs is to deal with exceptions which have been raised for the task. It is recommended that exceptions be reserved for errors and other abnormal conditions which arise.

A task's XSRs are activated asynchronously. This means that they are 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 the same priority as the task; it is only 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 before the next execution of the XSR will be seen as a single exception.

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 HIGH_PRIORITY. Priority HIGH_PRIORITY 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 creating task and can be changed later dynamically. They 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.

This standard defines four values for a mode parameter, and an **ORKID** compliant kernel may add others. A given mode is specified by a bit-field, similarly to events and exceptions. Each bit of a mode bit-field specifies a single mode value. The bit for each value is identified by a standard symbolic value - the mapping of these symbols to numeric values is implementation dependent. The four standard mode values are as follows:

+ NOXSR — This value affects only tasks with defined XSRs. When it is set, the task's XSRs will not be activated when exceptions are raised. Instead, exceptions will be latched until this value is cleared, after which the XSRs will be scheduled normally. Exceptions sent to a task without defined corresponding XSRs are lost.

+ NOTERMINATION — When this value is set, the task is protected from forced deletion or restart by other tasks. NOTERMINATION allows a task to complete a section of code without risk of deletion or restart, and yet still allows other tasks to be scheduled.

+ NOPREEMPT — When this value is set, the task will retain control of it's CPU either until it clears the value, or until it blocks itself by an **ORKID** operation call. In this latter case, when the task is eventually re-scheduled, the NOPREEMPT value will still be set in its mode. In this mode the task is also protected from being suspended by another task. This value does not preclude activation of XSRs or ISRs.

+ NOINTERRUPT — Tasks with this value set will not be interrupted.

Observation:

The NOINTERRUPT mode value does not preclude the execution of Interrupt Service Routines (ISR) by another processor in a multiple-processor node and therefor should not be used to obtain mutual exclusion with ISR code.

Observation:

A typical extension for certain processor architectures will be a SUPERVISOR mode value.

The behavior of a task is determined by the task's active mode. When a task is not executing an Exception Service Routine the mode specified in the task_create operation or the last task_set_mode operation is the active mode. Upon the activation of a task's XSR a new active mode is constructed by oring the old active mode with the mode specified in the exception_catch operation.
After returning to the interrupted task this one will continue in its old active mode (see also 10. Exceptions).

Observation:

An XSR should, in general, not reset any mode value via the task_set_mode operation that was set at the time of it's activation. This would lower the task's protection in an unforeseeable way.

Task Note-Pads

Every task has a fixed number of note-pad locations. These are simply 'word' locations which are accessible at all times by their own task, by all other tasks on the same node, and if the task was created with the GLOBAL option set, by all tasks on all nodes. The size of a note-pad location is equal to the basic word length of the corresponding processor. The note-pad is very simple, having only two operations -one to read and one to write a location.