2.6 ORKID Conformence

There are several places in this standard where the exact algorithms to be used are defined by the implementor of the compliant kernel. Although each operation has a defined functionality, the method used to achieve that functionality may cause behavioral differences.

For example, **ORKID** does not define the kernel scheduling algorithm, especially when several ready tasks have the same priority. This may lead to tasks being scheduled differently in different implementations, which may lead to possible different behavior.

Another example is the segment allocation algorithm. Different kernels may handle fragmentation in different ways, leading to cases where one implementation can fulfil a segment request, but another returns an error, since it has left the region more fragmented.

Subsets and Extensions

ORKID compliant kernels must implement all operations and objects as defined in this document; no subsets are permitted. Any **ORKID** compliant implementation may add extensions to give functionality in addition to that defined by this standard. Clearly, a task which uses non-standard extensions is unlikely to be portable to a standard system. In all cases, a kernel which claims compliance to **ORKID** should have all extensions clearly marked in its documentation.

Observation:

Hooks for user written extensions to the kernel will ease adaptation of ORKID compliant kernels to specific needs.

Undefined and Optional Items

There are several items which **ORKID** does not define but leaves up to the implementation.

ORKID does not define how system or node start-up is accomplished; this will obviously lead to differences in behavior, especially in multiple node systems.

ORKID does not define the word length. On this depends the size of integer parameters and bit-fields. These will be defined in the language binding along with all the other data structures, and so should not cause problems. It is envisaged that **ORKID** should be scalable - in other words it should be implementable on hardware with a different word length without loss of portability.

ORKID does not define the maximum number of task note-pad locations. The minimum number is sixteen.

ORKID does not define the range of priority values. But it defines the literal HIGH_PRIORITY to improve portability.

ORKID defines neither inter-kernel communication methods nor kernel

data structure implementations. This means that there is no requirement that one implementation must co-operate with other implementations within a system. In general, all the nodes in a system will run the same kernel implementation on nodes with the same integer size.

ORKID does not define whether object identifiers need be unique only at the current time, or must be unique throughout the system lifetime. A task which assumes the latter may have problems with an implementation which provides the former.

ORKID does not define the size limits on granularity for regions and buffer size for pools.

ORKID does not define any restrictions on the execution of operations within Interrupt Service Routines (ISRs). It does however define a minimum requirement of operations that must be supported.

ORKID defines a number of completion statuses. If an implementation does check for the condition corresponding to one of these statuses, then it must return the appropriate status.

ORKID does not define which completion status will be returned if multiple conditions apply.

ORKID does not define the encoding (binary value) of completion statuses, options and other symbolic values. But these values must be defined in the language binding.

ORKID does not define the maximum message length supported by a given implementation.

ORKID does not define the encoding of port designations in multi-port memory.

2.7. Layout of Operation Descriptions

The remainder of this standard is divided into one section per **ORKID** object type. Each section contains a detailed description of this type of object, followed by subsections containing descriptions of the relevant **ORKID** operations.

These operation descriptions are layed out in a formal manner, and contain information under the following headings:

Synopsis

This is a pseudo-language call to the operation giving its standard name and its list of parameters. Note that the language bindings define the actual names which are used for operations and parameters, but the order of the parameters in the call is defined here.

Input Parameters

Those parameters which pass data to the operation are given here in the format:

<parameter name> : <parameter type> commentary

The actual names to be used for parameters and their types are given definitively in the language bindings.

Output Parameters

Those parameters which return data from the operation are given here in the same format as for input parameters. Note that the types given here are simply the types of the data actually passed, and take no account of the mechanism whereby the data arrives back in the calling program. The actual parameter names and types to be used are given definitively in the language bindings.

Literal Values

Under this heading are given literal values which are used with given parameters. They are presented in the following two formats:

<parameter name> = <literal value> commentary
<parameter name> + <literal value> commentary

The first format indicates that the parameter is given exactly the indicated literal value if the parameters should affect the function desired in the commentary. The second format indicates that more than one such literal value for this parameter may be combined (logical or) and passed to or returned from the operation. If none of the defined conditions is set, the value of the parameter must be zero. The literal **ZERO** is defined in **ORKID** for initializing options and mode to this value.

Completion Status

Under this heading are listed all of the possible standard completion statuses that the operation may return.

Description

The last heading contains a description of the functionality of the operation. This description should not be interpreted as a recipe for implementation.

3. NODES

Nodes are the building bricks of **ORKID** systems, referenced by a node identifier and containing a single set of **ORKID** data structures. Nodes will typically contain a single CPU, but multi-CPU nodes are equally possible.

Specifying how nodes are created and configured is outside the scope of this standard. However, certain basic operations for node handling will be needed in all **ORKID** implementations and are defined in the following sections.

3.1. NODE_IDENT

Obtain the identifier of a node with a given name.

Synopsis

node_ident(name, nid)

Input Parameters

name	: string	user defined node name

Output Parameters

nid	: node_id	system defined node identifier

Literal Values

name	= WHO_AM_I	returns nid of calling task

Completion Status

OK	node_ident successful
ILLEGAL_USE	node_ident not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
NAME_NOT_FOUND	no node with this name

Description

This operation returns the node identifier for the node with the given name. No assumption is made on how this identifier is obtained. It there is more than one mode with the same name in the system, then the nid of the first one found is returned.