Open Real-time Kernel Interface Definition Page 95

Draft 2.1

queue\_delete ( qid )

queue\_ident ( name, nid, qid )

queue\_send ( qid, msg\_buff, msg\_length )

queue\_jump ( qid, msg\_buff, msg\_length )

queue\_broadcast ( qid, msg\_buff, msglength, count )

queue\_receive ( qid, msg\_buff, buff\_length, options, time\_out,

msg length )

queue\_flush ( qid, count )

queue\_info ( qid, max\_buff, length, options, messages\_waiting,

tasks\_waiting )

**Event Operations**

event\_send ( tid, event )

event\_receive ( event, options, time\_out, event\_received )

**Exception Operations**

exception\_catch ( bit\_number, new\_xsr, new\_mode, old\_xsr, old\_mode)

exception\_raise ( tid, exception )

exception\_return ( )

**Clock Operations**

clock\_set ( clock )

clock\_get ( clock )

clock\_tick ( )

**Timer Operations**

timer\_wake\_after ( ticks )

timer\_wake\_when ( clock )

timer\_event\_after ( ticks, event, tmid )

timer\_event\_when ( clock, event, tmid )

timer\_event\_every ( ticks )

timer\_cancel ( tmid )

**Interrupt Operations**

int\_enter ( )

int\_return ( )

**Miscellaneous Operations**

int\_to\_ext ( int\_addr, port, ext\_addr )

ext\_to\_int ( ext\_addr, port, int\_addr )

Open Real-time Kernel Interface Definition Page 96

Draft 2.1

#ifndef ORKID\_H

#define ORKID\_H 1

/\*

**D. ORKID: C LANGUAGE BINDING**

This file contains the C language binding standard for VITA's "Open

Real-time Kernel Interface Definition", henceforth called ORKID. The

file is in the format of a C language header file, and is intended to be

a common starting point for system developers wishing to produce an

ORKID compliant kernel.

The ORKID C language binding consists of four sections, containing type

specifications, function declarations, completion status codes and

special symbol codes. The character sequence ??? has been used

throughout wherever the coding is implementation dependent.

Of the four sections in this standard, only the function declarations

are completely defined. In the other sections, only the type names and

constant symbols are defined by this standard - all types and values are

implementation dependent.

Both ANSI C and non-ANSI C have been used for this header file.

Defining the symbol ANSI will cause the ANSI versions to be used,

otherwise the non-ANSI versTons will be used. Full prototyping has been

employed for the ANSI function declarations.

\*/

Open Real-time Kernel Interface Definition Page 97

Draft 2.1

/\*

**ORKID TYPE SPECIFICATIONS**

This section of the ORKID C language binding contains typedef

definitions for the types used in operation arguments in the main ORKID

standard. The names are the same as those in the ORKID standard. Only

the names, and in clock\_buff the order of the structure members, are

defined by this standard. The actual types are implementation dependent.

\*/

typedef unsigned int prio ;

typedef unsigned int word ;

typedef unsigned int bit\_field ;

typedef ??? task\_id ;

typedef ??? node\_id ;

typedef ??? region\_id ;

typedef ??? pool\_id ;

typedef ??? sema\_id ;

typedef ??? queue\_id ;

typedef ??? timer\_id ;

typedef ??? cb\_year ;

typedef ??? cb\_month ;

typedef ??? cb\_day ;

typedef ??? cb\_hours ;

typedef ??? cb\_minutes ;

typedef ??? cb\_seconds ;

typedef ??? cb\_ticks ;

typedef ??? cb\_time\_zone ;

typedef ??? clock\_buff ;

Open Real-time Kernel Interface Definition Page 98

Draft 2.1

/\*

**ORKID OPERATION DECLARATIONS**

This section of the ORKID C language binding contains function

declarations for all the operations defined in the main ORKID standard,

and is subdivided according to the subsections in this standard.

Each subdivision contains a list of function declarations and a list of

symbol definitions. The function names have been kept to six characters

for the sake of linker compatibility. Of these six characters, the

first two are always 'OK', and the third designates the ORKID object

type on which the operation works. The symbol definitions link the full

names of the operations given in the ORKID standard (in lower case) to

the appropriate abbreviation.

The lists of function declarations are split in two. If the symbol

\_\_ANSI\_\_ has been defined, then all the functions are declared to the

ANSI C standard using full prototyping, with parameter names also

included. This latter is not necessary, but not illegal. It shows the

correspondence between arguments in this and the main ORKID standard,

the names being identical. If the symbol \_\_ANSI\_\_ has not been defined,

then the functions are declared without prototyping.

The correspondence between the C types and arguments and those defined

in the ORKID standard are mostly obvious. However, the following

comments concerning task\_start/restart and exception\_catch are perhaps

necessary.

A task start address is translated into a function with one argument -a

pointer to anything. The task's startup arguments are given as a pointer

to anything and a length. The actual arguments will be contained in a

programmer defined data type, a copy of which will be passed to the new

task. The following is an example of a declaration of a task's main

program and a call to start that task (the necessary task creation call

is not included):

typedef struct { int argl, arg2, arg3 } argblock ; /\*can contain

argblock \*argp ; anything\*/

void taskmain( argblock \*taskargs, int arg\_size ) {...}; /\*main

task program\*/

status = oktsta( tid, taskmain, \*argp, size\_of( argblock ) ) ;

/\*start the task\*/

Open Real-time Kernel Interface Definition Page 99

Draft 2.1

An XSR address also becomes a function with one argument - this time a

bitfield. The previous XSR address output parameter becomes a pointer

to such a function. The following is an example of the declaration of

an XSR and a call to exception\_catch to set it up:

void taskxsr( bit\_field exception\_caught ){...}; /\*XSR

declaration\*/

void (\*oldxsr) () ;

status = okxcat( taskxsr, NOXSR, oldxsr ) ; /\*set up taskxsr as XSR\*/

with NOXSR mode parameter

\*/

Open Real-time Kernel Interface Definition Page 100

Draft 2.1

**/\* Task Operations \*/**

#ifdef \_\_ANSI\_\_

extern int oktcre( char \*name, prio priority, int stacksize, bit\_field

mode, bit\_field options, task\_id tid ) ;

extern int oktdel( task\_id tid ) ;

extern int oktidt( char \*name, node id nid, task\_id tid ) ;

extern int oktsta( task\_id tid, void start(void \*), void \*arguments,

int arg\_length ) ;

extern int oktrst( task\_id tid, void \*arguments, int arg\_length ) ;

extern int oktsus( task\_id tid ) ;

extern int oktrsm( task\_id tid ) ;

extern int oktspr( task\_id tid, prio new\_prio, prio \*old\_prio ) ;

extern int oktsmd( bit\_field new\_mode, bit\_field mask, bit\_field

\*old\_mode ) ;

extern int oktrnp( task\_id tid, int loc\_number, word \*loc\_value ) ;

extern int oktwnp( task\_id tid, int loc\_number, word loc\_value ) ;

extern int oktinf( task\_id tid, prio \*priority, bit\_field \*mode,

bit\_field \*options, bit\_field \*event, bit\_field

\*exception, int state );

#else

extern int oktcre( ) ;

extern int oktdel( ) ;

extern int oktidt( ) ;

extern int oktsta( ) ;

extern int oktrst( ) ;

extern int oktsus( ) ;

extern int oktrsm( ) ;

extern int oktspr( ) ;

extern int oktsmd( ) ;

extern int oktrnp( ) ;

extern int oktwnp( ) ;

extern int oktinf( ) ;

#endif

#define task\_create oktcre

#define task\_delete oktdel

#define task\_ident oktidt

#define task\_start oktsta

#define task\_restart oktrst

#define task\_suspend oktsus

#define task\_resume oktrsm

#define task\_set\_priority oktspr

#define task\_set\_mode oktsmd

#define task\_read\_note\_pad oktrnp

#define task\_write\_note\_pad oktwnp

#define task info oktinf

Open Real-time Kernel Interface Definition Page 101

Draft 2.1

**/\* Region Operations \*/**

#ifdef \_\_ANSI\_\_

extern int okrcre( char \*name, void \*addr, int length, int granularity,

bit\_field options, region\_id \*rid ) ;

extern int okrdel( region\_id rid ) ;

extern int okridt( char \*name, region\_id \*rid ) ;

extern int okrgsg( region\_id rid, int seg\_size, void \*\*seg\_addr ) ;

extern int okrrsg( region\_id rid, void \*seg\_addr ) ;

extern int okrinf( region\_id rid, int size, int max\_segment,

int granularity, bit\_field options)

#else

extern int okrcre( ) ;

extern int okrdel( ) ;

extern int okridt( ) ;

extern int okrgsg( ) ;

extern int okrrsg( ) ;

extern int okrinf( ) ;

#endif

#define region\_create okrcre

#define region\_delete okrdel

#define region\_ident okridt

#define region\_get\_seg okrgsg

#define region\_ret\_set okrrsg

#define region\_info okrinf

Open Real-time Kernel Interface Definition Page 102

Draft 2.1

**/\* Pool Operations \*/**

#ifdef \_\_ANSI\_\_

extern int okpcre( char \*name, void \*addr, int length, int block\_size,

bit\_field options, pool\_id \*pid ) ;

extern int okpdel( pool\_id pid ) ;

extern int okpidt( char \*name, node\_id nid, pool\_id \*pid);

extern int okpgbl( pool\_id pid, void \*\*blk\_addr ) ;

extern int okprbl( pool\_id pid, void \*blk\_addr ) ;

extern int okpinf( pool\_id pid, int buffers, int free\_buffers,

int buff\_size, bit\_field options)

#else

extern int okpcre( ) ;

extern int okpdel( ) ;

extern int okpidt( ) ;

extern int okpgbl( ) ;

extern int okprbl( ) ;

extern int okpinf( ) ;

#endif

#define pool\_create okpcre

#define pool\_delete okpdel

#define pool\_ident okpidt

#define pool\_get\_blk okpgbl

#define pool\_ret\_blk okprbl

#define pool\_info okpinf

Open Real-time Kernel Interface Definition Page 103

Draft 2.1

**/\* Semaphore Operations \*/**

#ifdef \_\_ANSI\_\_

extern int okscre( char \*name, int init\_count, bit\_field options, sem\_id

\*sid ) ;

extern int oksdel( sem\_id \*sid ) ;

extern int oksidt( char \*name, node\_id nid, sem\_id \*sid ) ;

extern int okstak( sem\_id \*sid, bit\_field options, int time\_out ) ;

extern int okssig( sem\_id \*sid ) ;

extern int oksinf( sem\_id \*sid, bit\_field options, int count,

int tasks\_waiting)

#else

extern int okscre( ) ;

extern int oksdel( ) ;

extern int oksidt( ) ;

extern int okstak( ) ;

extern int okssig( ) ;

extern int oksinf( ) ;

#endif

#define sem\_create okscre

#define sem\_delete oksdel

#define sem\_ident oksidt

#define sem\_take okstak

#define sem\_signal okssig

#define sem\_info oksinf

Open Real-time Kernel Interface Definition Page 103

Draft 2.1

**/\* Queue Operations \*/**

#ifdef \_\_ANSI\_\_

extern int okqcre( char \*name, int max\_buff, int length,

bit\_field options, queue\_id \*gid ) ;

extern int okqdel( queue\_id qid ) ;

extern int okqidt( char \*name, node\_id nid, queue\_id \*qid ) ;

extern int okqsnd( queue\_id qid, void \*msg\_buff, int msg\_length ) ;

extern int okqjmp( queue\_id qid, void \*msg\_buff, int msg\_length );

extern int okqbro( queue\_id qid, void \*msg\_buff, int msg\_length,

int \*count ) ;

extern int okqrcv( queue\_id qid, void \*msg\_buff, int buff\_length,

bit\_field options, int time\_out, int length );

extern int okqflu( queue\_id qid, int \*count ) ;

extern int okqinf( queue\_id qid, int max\_buff, int length,

bit\_field options, int messages\_waiting,

int tasks\_waiting)

#else

extern int okqcre( ) ;

extern int okqdel( ) ;

extern int okqidt( ) ;

extern int okqsnd( ) ;

extern int okqbro( ) ;

extern int okqjmp( ) ;

extern int okqrcv( ) ;

extern int okqflu( ) ;

extern int okqinf( ) ;

#endif

#define queue\_create okqcre

#define queue\_delete okqdel

#define queue\_ident okqidt

#define queue\_send okqsnd

#define queue\_broadcast okqbro

#define queue\_jump okqjmp

#define queue\_receive okqrcv

#define queue\_flush okqflu

#define queue\_info okqinf