12. INTERRUPTS

ORKID defines two operations which bracket interrupt handler code. It is up to each implementor to decide what functionality, to put in these operations.

Observation:

The kernel may use int_enter and int_exit with an Interrupt Service Routine code or task code is being executed. Typically int_exit will be used to decide if a scheduling action must take place in pre-emptive kernels.

12.1. INT_ENTER

Announce interrupt handler entry.

Synopsis

```
   int_enter( )
```

Input Parameters

   <none>

Output Parameters

   <none>

Completion Status

OK	int_enter operation successful

Description

This operation call announces the start of an interrupt handling routine to the kernel. Its functionality is implementation dependent. The operation takes no parameters and always returns a successful completion status. It is up to a user task to set up vectors to the handler which makes this call.

12.2. INT_EXIT

Exit from an interrupt handler.

Synopsis

```
int_exit( )
```

Input Parameters

   <none>

Output Parameters

   <none>

Completion Status

   <not applicable>

Description

This operation announces the end of an interrupt handling routine to the kernel. Its exact functionality is implementation dependent, but will involve returning to interrupted code or scheduling another task. The operation takes no parameters and does not return to the calling code.

The behavior of int_return when not called from an Interrupt Service Routine is undefined.

A. RETURN CODES

CLOCK_NOT_SET	clock has not been initialized
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID OPTIONS	invalid options value
INVALID_ADDRESS	a specific parameter refers to an illegal address
INVALID_ARGUMENTS	invalid number or type or size of arguments
INVALID_BLOCK	no block allocated from partition at blk_addr
INVALID_BLOCK_SIZE	block_size not supported
INVALID_CLOCK	invalid clock value
INVALID_COUNT	init count is negative
INVALID_GRANULARITY	granularity not supported
INVALID_ID	object does not exist
INVALID_LENGTH	buffer length not supported
INVALID_LOCATION	note-pad number does not exist
INVALID_MODE	invalid mode or mask value
INVALID_NODE	node does not exist
INVALID_OPTIONS	invalid options value
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_PRIORITY	invalid priority value
INVALID_SEGMENT	no segment allocated from this region at seg_addr
NAME_NOT_FOUND	name does not exist on node
NODE_NOT_REACHABLE	node on which object resides is not reachable
NO_EVENTS	event(s) not set and NOWAIT option given
NO_MORE_MEMORY	not enough memory to satisfy request
OBJECT_DELETED	specified object has been deleted
OBJECT_PROTECTED	task has NOPREEMPT or NOTERMINATION parameter set
OK	operation successful
PARTITION_IN_USE	blocks from this partition are still allocated
PARTITION_OVERLAP	Area given overlaps an existing partition
QUEUE_DELETED	queue deleted while blocked in queue_receive operation
QUEUE_EMPTY	queue empty with NOWAIT option
QUEUE_FULL	no more buffers available
REGION_IN_USE	segments from this region are still allocated
REGION_OVERLAP	area given overlaps an existing region
SEMAPHORE_DELETED	semaphore deleted while blocked in sem_p operation
SEMAPHORE_NOT_AVAILABLE	semaphore unavailable with NOWAIT option
SEM_OVERFLOW	the counter of semaphore overflows
TASK_ALREADY_STARTED	task has been started already
TASK_ALREADY_SUSPENDED	task already suspended
TASK_NOT_STARTED	task has not yet been started
TASK_NOT_SUSPENDED	task not suspended
TIME_OUT	operation timed out
TOO_MANY_PARTITIONS	too many partitions on the node
TOO_MANY_QUEUES	too many queues on node
TOO_MANY_REGIONS	too many regions on the node
TOO_MANY_SEMAPHORES	too many semaphores on node
TOO_MANY_TASKS	too many tasks on the node
TOO_MANY_TIMERS	too many timers on node
XSR_NOT_SET	task has no exception handler routine

B. MINIMUM REQUIREMENTS FOR OPERATIONS FROM AN ISR.

ORKID requires that at least the following operations are supported from an Interrupt Service Routine. Only operations on local objects need to be supported. If the object resides on a remote node and remote operations are not supported, then the INVALID_ID completion status must be returned.

Task Operations

task_suspend	(tid)
task_resume	(tid)
task_read_notepad	(tid, loc_number, loc_value)
task_write_notepad	(tid, loc_number, loc_value)

Semaphore Operations

sem_v	(sid)

Queue Operations

queue_send	(qid, message, length)
queue_urgent	(qid, message, length)

Event Operations

event_send	(tid, event)

Exception Operations

exceptions_raise	(tid, exceptions)

Clock Operations

clock_tick	( )	clock_get	(clock)

Interrupt Operations

int_enter	( )
int_exit	( )

C. MINIMUM REQUIREMENTS FOR OPERATIONS FROM AN XSR.

ORKID requires that at least the following operations are supported from an Exception Service Routine.

Task Operations

task_delete	(tid)
task_start	(tid, start_addr, arguments)
task_restart	(tid, arguments)
task_suspend	(tid)
task_resume	(tid)
task_set_priority	(tid, new_prio, old_prio)
task_set_mode	(mode, mask, old_mode)
task_read_notepad	(tid, loc_number, loc_value)
task_write_notepad	(tid, loc_number, loc_value)

Region Operations

region_delete	(rid, options)
region_get_seg	(rid, seg_size, seg_addr)
region_ret_seg	(rid, seg_addr)
region_info	(rid, size, max_segment, granularity)

Partition Operations

partition_delete	(pid, options)
partition_get_blk	(pid, blk_addr)
partition_ret_blk	(pid, blk_addr)
partition_info	(pid, blocks, free_blocks, block_size)

Semaphore Operations

sem_delete	(sid)
sem_p	(sid, time_out)
sem_v	(sid)
sem_info	(sit, options, count, tasks_waiting)

Queue Operations

queue_delete	(qid)
queue_send	(qid, message, length)
queue_urgent	(qid, message, length)
queue_broadcast	(qid, message, length, count)
queue_receive	(qid, message, time_out)
queue_flush	(qid, count)
queue_info	(qid, max_buf, length, options, messages_waiting, tasks_waiting)

Event Operations

event_send	(tid, event)
event_receive	(events, options, time_out, events_caught)

Exception Operations

```
exceptions_send      ( tid, exceptions )
exceptions_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_cancel         ( tmid )
```

D. SUMMARY OF ORKID OPERATIONS

In the following summary, output parameters are underlined.

Task Operations

```
task_create         ( name, priority, stack_size, mode, options, tid )
task_delete         ( tid )
task_ident          ( name, nid, tid )
task_start          ( tid, start_addr, arguments )
task_restart        ( tid, arguments )
task_suspend        ( tid )
task_resume         ( tid )
task_set_priority   ( tid, new_prio, old_prio )
task_set_mode       ( mode, mask, old_mode )
task_read_notepad   ( tid, loc_number, loc_value )
task_write_notepad  ( tid, loc_number, loc_value )
```

Region Operations

```
region_create       ( name, addr, length, granularity, options, rid )
region_delete       ( rid, options )
region_ident        ( name, rid )
region_get_seg      ( rid, seg_size, seg_addr )
region_ret_seg      ( rid, seg_addr )
region_info         ( rid, size, max_segment, granularity )
```

Partition Operations

```
partition_create    ( name, addr, length, block_size, options, pid )
partition_delete    ( pid, options )
partition_ident     ( name, nid, pid, block_size )
partition_get_blk   ( pid, blk_addr )
partition_ret_blk   ( pid, blk_addr )
partition_info      ( pid, blocks, free_blocks, block_size )
```

Semaphore Operations

```
sem_create          ( name, count, options, sid )
sem_delete          ( sid )
sem_ident           ( name, nid, sid )
sem_p               ( sid, time_out )
sem_v               ( sid )
sem_info            ( sit, options, count, tasks_waiting )
```

Queue Operations

```
queue_create        ( name, priv_buff, max_buff, length, options, qid )
queue_delete        ( qid )
queue_ident         ( name, nid, qid )
queue_send          ( qid, message, length )
queue_urgent        ( qid, message, length )
queue_broadcast     ( qid, message, length, count )
queue_receive       ( qid, message, time_out )
```

queue_flush	(qid, count)
queue_info	(qid, max_buf, length, options, messages_waiting, tasks_waiting)

Event Operations

event_send	(tid, event)
event_receive	(events, options, time_out, events_caught)

Exception Operations

exceptions_catch	(new_XSR, mode, old_XSR, old_mode)
exceptions_send	(tid, exceptions)
exceptions_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_cancel	(tmid)

Interrupt Operations

int_enter	( )
int_exit	( )

```
#ifndef ORKID_H
#define ORKID_H 1
/*
```

E. 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. Nevertheless, where possible, example values have been given.

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 versions will be used. Full prototyping has been employed for the ANSI function declarations.

```
*/
```

/*

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_buf the order of the structure members, are defined by this standard. The actual types are implementation dependent.
*/

```
typedef unsigned int prio ;
typedef unsigned int lnum ;
typedef unsigned int bit_field ;
typedef struct { ??? } task_id ;
typedef struct { ??? } node_id ;
typedef struct { ??? } region_id ;
typedef struct { ??? } part_id ;
typedef struct { ??? } sema_id ;
typedef struct { ??? } queue_id ;
typedef struct { ??? } timer_id ;
typedef struct {
     ??? cb_year ;
     ??? cb_month ;
     ??? cb_day ;
     ??? cb_hours ;
     ??? cb_minutes ;
     ??? cb_seconds ;
     ??? cb_ticks ;
     ??? cb_time_zone ; } clock_buf ;
```

/*

ORKID OPERATION DECLARATIONS

This section of the the **ORKID** C language binding is the largest and 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 arg1, arg2, arg3 } argblock ; /*can contain
argblock *argp ;                                     anything*/


void taskmain( argblock *taskargs ) { ... } ;      /*main task program*/


status = oktsta( tid, taskmain, *argp, size_of( argblock ) ) ;

/*start the task*/
```

An XHR address also becomes a function with one argument - this time a bitfield. The previous XHR address output parameter becomes a pointer to such a function. The following is an example of the declaration of an XHR and a call to exception_catch to set it up:

```
void taskxhr( bit_field exceptions_caught ) { ... } ;  /*XHR
                                                        declaration*/
void (*prevxhr)() ;

status = okxcat( taskxhr, NOXHR, prevxhr ) ;  /*set up taskxhr as XHR*/
*/
```

```
/*    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 node, 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 *prev_prio ) ;
extern int oktsmd( bit_field mode, bit_field mask, bit_field
                   *prev_mode ) ;
extern int oktrdl( task_id *tid, lnum loc_number, int *loc_value ) ;
extern int oktwrl( task_id *tid, lnum loc_number, int loc_value ) ;

#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 oktrdl( ) ;
extern int oktwrl( ) ;

#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_location    oktrdl
#define task_write_location   oktwrl
```

```
/*    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, bit_field options ) ;
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 ) ;

#else

extern int okrcre( ) ;
extern int okrdel( ) ;
extern int okridt( ) ;
extern int okrgsg( ) ;
extern int okrrsg( ) ;

#endif

#define region_create     okrcre
#define region_delete     okrdel
#define region_ident      okridt
#define region_get_seg    okrgsg
#define region_ret_set    okrrsg
```

```
/*    Partition Operations    */


#ifdef __ANSI__

extern int okpcre( char *name, void *addr, int length, int block_size,
                   bit_field options, part_id *pid ) ;
extern int okpdel( part_id *pid, bit_field options ) ;
extern int okpidt( char *name, node_id nid, part_id *pid,
                   int block_size ) ;
extern int okpgbl( part_id *pid, void **blk_addr ) ;
extern int okprbl( part_id *pid, void *blk_addr ) ;

#else

extern int okpcre( ) ;
extern int okpdel( ) ;
extern int okpidt( ) ;
extern int okpgbl( ) ;
extern int okprbl( ) ;

#endif

#define partition_create      okpcre
#define partition_delete      okpdel
#define partition_ident       okpidt
#define partition_get_blk     okpgbl
#define partition_ret_blk     okprbl
```

```
/*    Semaphore Operations    */

#ifdef __ANSI__

extern int okscre( char *name, int count, bit_field options, sema_id
                   *sid ) ;
extern int oksdel( sema_id *sid ) ;
extern int oksidt( char *name, node_id nid, sema_id *sid ) ;
extern int oksemp( sema_id *sid, int time_out ) ;
extern int oksemv( sema_id *sid ) ;

#else

extern int okscre( ) ;
extern int oksdel( ) ;
extern int oksidt( ) ;
extern int oksemp( ) ;
extern int oksemv( ) ;

#endif

#define sem_create  okscre
#define sem_delete  oksdel
#define sem_ident   oksidt
#define sem_p       oksemp
#define sem_v       oksemv
```

```
/*    Queue Operations    */


#ifdef __ANSI__

extern int okqcre( char *name, int priv_buff, int max_buff, int length,
                   bit_field options, queue_id *qid ) ;
extern int okqdel( queue_id *qid ) ;
extern int okqidt( char *name, node_id nid, queue_id *qid ) ;
extern int okqsnd( queue_id *qid, void *message, int length ) ;
extern int okqurg( queue_id *qid, void *message, int length ) ;
extern int okqbro( queue_id *qid, void *message, int length,
                   int *count ) ;
extern int okqrcv( queue_id *qid, void *message, int time_out ) ;
extern int okqflu( queue_id *qid, int *count ) ;

#else

extern int okqcre( ) ;
extern int okqdel( ) ;
extern int okqidt( ) ;
extern int okqsnd( ) ;
extern int okqurg( ) ;
extern int okqbro( ) ;
extern int okqrcv( ) ;
extern int okqflu( ) ;

#endif

#define queue_create     okqcre
#define queue_delete     okqdel
#define queue_ident      okqidt
#define queue_send       okqsnd
#define queue_urgent     okqurg
#define queue_broadcast  okqbro
#define queue_receive    okqrcv
#define queue_flush      okqflu
```

```
/*    Event Operations    */

#ifdef __ANSI__

extern int okesnd( task_id *tid, bit_field event ) ;
extern int okercv( bit_field events, bit_field options, int timeout,
                   bit_field *events_caught ) ;

#else

extern int okesnd( ) ;
extern int okercv( ) ;

#endif


#define event_send         okesnd
#define event_receive      okercv
```

```
/*    Exception operations     */


#ifdef __ANSI__

extern int okxcat( void new_xhr(bit_field), bit_field mode,
                   void (*old_xhr)(bit_field), bit_field *old_mode ) ;
extern int okxsnd( task_id *tid, bit_field exceptions ) ;
extern void okxret( void ) ;

#else

extern int okxcat( ) ;
extern int okxsnd( ) ;
extern void okxret( ) ;

#endif


#define exceptions_catch       okxcat
#define exceptions_send        okxsnd
#define exceptions_return      okxret
```

```
/*    Clock Operations    */


#ifdef __ANSI__

extern int okcset( clock_buf *clock ) ;
extern int okcget( clock_buf *clock ) ;
extern int okctik( void ) ;

#else

extern int okcset( ) ;
extern int okcget( ) ;
extern int okctik( ) ;

#endif

#define clock_set   okcset
#define clock_get   okcget
#define clock_tick  okctik
```

```
/*    Timer Operations     */


#ifdef __ANSI__

extern int oktmwa( int ticks ) ;
extern int oktmww( clock_buf clock ) ;
extern int oktmea( int ticks, bit_field event, timer_id *tmid ) ;
extern int oktmew( clock_buf clock, bit_field event, timer_id *tmid ) ;
extern int oktcan( timer_id *tmid ) ;

#else

extern int oktmwa( ) ;
extern int oktmww( ) ;
extern int oktmea( ) ;
extern int oktmew( ) ;
extern int oktcan( ) ;

#endif

#define timer_wake_after       oktmwa
#define timer_wake_when        oktmww
#define timer_event_after      oktmea
#define timer_event_when       oktmew
#define timer_cancel           oktmca
```

```
/*    Interrupt Operations    */

#ifdef __ANSI__

extern int okient( void ) ;
extern void okiexi( void ) ;

#else

extern int okient( ) ;
extern void okiexi( ) ;

#endif


#define int_enter   okient
#define int_exit    okiexi
```

/*

COMPLETION STATUS CONSTANTS

This section of the **ORKID** C language binding contains definitions for all the completion status values used in the main **ORKID** standard. The **symbols** used are the same as those given in the main standard, and are defined for C by this standard. Of the values, only the value 0 for the completion status 'OK' is defined here - the other values are given only as examples.
*/

```
#define OK                         0
#define CLOCK_NOT_SET              1
#define COUNT_TOO_HIGH             2
#define ILLEGAL_USE                3
#define INVALID_ADDRESS            4
#define INVALID_ARGUMENT           5
#define INVALID_BLOCK              6
#define INVALID_BLOCK_SIZE         7
#define INVALID_CLOCK              8
#define INVALID_COUNT              9
#define INVALID_GRANULARITY       10
#define INVALID_ID                11
#define INVALID_LENGTH            12
#define INVALID_LOCATION          13
#define INVALID_MAX_BUFF          14
#define INVALID_MODE              15
#define INVALID_NAME              16
#define INVALID_NODE              17
#define INVALID_OPTIONS           18
#define INVALID_PRIORITY          19
#define INVALID_SEGMENT           20
#define NAME_NOT_FOUND            21
#define NO_EVENTS                 22
#define NO_MORE_MEMORY            23
#define NODE_NOT_REACHABLE        24
#define OBJECT_DELETED            25
#define OBJECT_NOT_GLOBAL         26
#define PARTITION_IN_USE          27
#define PARTITION_OVERLAP         28
#define QUEUE_DELETED             29
#define QUEUE_EMPTY               30
#define QUEUE_FULL                31
#define REGION_IN_USE             32
#define REGION_OVERLAP            33
#define SEMAPHORE_DELETED         34
#define SEMAPHORE_NOT_AVAILABLE   35
#define TASK_ALREADY_STARTED      36
#define TASK_ALREADY_SUSPENDED    37
#define TASK_MARKED_FOR_DELETE    38
#define TASK_MARKED_FOR_RESTART   39
#define TASK_NOT_SUSPENDED        40
#define TIME_OUT                  41
#define TOO_MANY_PARTITIONS       42
```

```
#define TOO_MANY_QUEUES             43
#define TOO_MANY_REGIONS            44
#define TOO_MANY_SEMAPHORES         45
#define TOO_MANY_TASKS              46
#define TOO_MANY_TIMERS             47
#define XHR_NOT_SET                 48
```

```
/*


LITERAL VALUES

This section of the ORKID C language binding contains definitions for
all special symbols used in argument values in the main ORKID standard.
The symbols used are the same as those given in the main standard, and
are defined for C by this standard.  None of the values given here are
defined by this standard -  they are included as examples only.
*/

#define SELF                0           /* tid */

#define LOCAL_NODE          0           /* nid */
#define OTHER_NODES        -1

#define CURRENT             0           /* new_prio */
#define HIGHP               63          /* new_prio, prev_prio, priority */

#define NOXHR               0x1         /* mode, mask, prev_mode */
#define NOTERMINATION       0x2
#define NOPREEMPT           0x4
#define NOINTERRUPT         0x8

#define GLOBAL              0x0001      /* options */
#define FORCED_DELETE       0x0002
#define FIFO                0x0004
#define ANY                 0x0008
#define NOWAIT              0x0010

#define FOREVER             0           /* time_out */

#define NULL_XHR            0           /* new_xhr, prev_xhr */

#endif
```