13.1. INT_ENTER

Announce Interrupt Service Routine entry.

Synopsis

```
int_enter( )
```

Input Parameters

<none>

Output Parameters

<none>

Completion Status

OK	int_enter successful

Description

This operation announces the start of an Interrupt Service 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.

13.2. INT_RETURN

Exit from an Interrupt Service Routine

Synopsis

int_return ( )

Input Parameters

<none>

Output Parameters

<none>

Completion Status

<not applicable>

Description

This operation announces the return from an ISR 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 ISR is undefined.

14. MISCELLANEOUS

This chapter contains the descriptions of miscellaneous operations.

In the current revision of **ORKID** these are restricted to address translation operations. These operations translate addresses of multi-ported memory from local processor addresses to the corresponding addresses on other ports and vice-versa.

14.1. INT_TO_EXT

Translate processor address to external port address.

Synopsis

```
int_to_ext( int_addr, port, ext_addr )
```

Input Parameters

int_addr	: address	processor address to be translated
port	: integer	port designation

Output Parameters

ext_addr	: address	correspondig address for designated port

Completion Status

OK	int_to_ext successful
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_PORT	port does not exist
NO_TRANSLATION	int_addr can not be accessed through port

Description

This operation translates a processor address of a multi-port memory location to the address accessing the same location via the given port. The port parameter encodes the bus and address space to be used, e.g. VMEbus with a certain address modifier. If the given port does not exist the INVALID_PORT completion status is returned. If the given location cannot be accessed via the port the NO_TRANSLATION completion status is returned.

Observation:

It is assumed that the various bus standard authorities will define literals for the encoding of ports for their respective bus architectures.

14.2. EXT_TO_INT

Translate external port address to processor address.

Synopsis

```
ext_to_int( ext_addr, port, int_addr )
```

Input Parameters

ext_addr	: address	port address to be translated
port	: integer	port designation

Output Parameters

int_addr	: address	correspondig processor address

Completion Status

OK	ext_to_int successful
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_PORT	port does not exist
NO_TRANSLATION	ext_addr can not be accessed by processor

Description

This operation translates an external port address of a multi-port memory to the processor address accessing the same location. The port parameter encodes the bus and address space to be used, e.g. VMEbus with a certain address modifier. If the given port does not exist the INVALID_PORT completion status is returned. If the given location can not be accessed by the processor the NO_TRANSLATION completion status is returned (see also 14.1. Observation).

A. COMPLETION STATUSES

CLOCK_NOT_SET	clock has not been initialized
ILLEGAL_USE	operation not callable from ISR
INVALID_ARGUMENTS	invalid number or type or size of arguments
INVALID_BIT	invalid exception bit-number
INVALID_BUFF	no buffer allocated from partition at buff_addr
INVALID_BUFF_SIZE	buff_size not supported
INVALID_CLOCK	invalid clock value
INVALID_COUNT	initial 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_OPTIONS	invalid options value
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_PRIORITY	invalid priority value
INVALID_SEGMENT	no segment allocated from this region at seg_addr
NAME_NOT_FOUND	object name does not exist on node
NODE_NOT_REACHABLE	node on which object resides is not reachable
NO_EVENT	event(s) not set and NOWAIT option given
NO_MORE_MEMORY	not enough memory to satisfy request
OBJECT_DELETED	originally existing task has been deleted before operation
OBJECT_NOT_LOCAL	operation not allowed on non-local object
OBJECT_PROTECTED	task in NOTERMINATION mode
OK	operation successful
POOL_IN_USE	buffers from this pool are still allocated
POOL_NOT_SHARED	pool not in shared memory subsystem
POOL_OVERLAP	area given overlaps an existing pool
QUEUE_DELETED	queue deleted while blocked in queue_receive
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_claim
SEMAPHORE_NOT_AVAILABLE	semaphore unavailable with NOWAIT option
SEMAPHORE_OVERFLOW	semaphore counter overflowed
SEMAPHORE_UNDERFLOW	semaphore counter underflowed
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_OBJECTS	too many objects of given type on the node or in the system
XSR_NOT_SET	no handler routine for given exception(s)

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 OBJECT_NOT_LOCAL completion status must be returned.

Observation:

The SELF literal is meaningless for ORKID operations called from an ISR and will lead to the INVALID_ID completion status.

NODE OPERATIONS

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

Task Operations

```
task_suspend         ( tid )
task_resume          ( tid )
task_read_note-pad   ( tid, loc_number, loc_value )
task_write_note-pad  ( tid, loc_number, loc_value )
```

Semaphore Operations

```
sem_release           ( sid )
```

Queue Operations

```
queue_send           ( qid, msg_buff, msg_length )
queue_jump           ( qid, msg_buff, msg_length )
```

Event Operations

```
event_send           ( tid, event )
```

Exception Operations

```
exception_raise     ( tid, exception )
```

Clock Operations

```
clock-get            ( clock )
clock-tick           ( )
```

Interrupt Operations

```
int_enter            ( )
int_return           ( )
```

C. SUMMARY OF ORKID OPERATIONS

In the following, output parameters are printed in bold characters.

Node Operations

```
node_ident          ( name, nid )
node_fail           ( nid, code, options )
node_info           ( nid, ticks_per_sec )
```

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       ( new_mode, mask, old_mode )
task_read_note_pad  ( tid, loc_number, loc_value )
task_write_note_pad ( tid, loc_number, loc_value )
task_info           ( tid, priority, mode, options, event, exception )
```

Region Operations

```
region_create       ( name, addr, length, granularity, options, rid )
region_delete       ( rid )
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, options )
```

Pool Operations

```
pool_create         ( name, addr, length, buff_size, options, pid )
pool_delete         ( pid )
pool_ident          ( name, nid, pid)
pool_get_buff       ( pid, buff_addr )
pool_ret_buff       ( pid, buff_addr )
pool_info           ( pid, buffers, free_buffers, buff_size, options )
```

Semaphore Operations

```
sem_create          ( name, init_count, options, sid )
sem_delete          ( sid )
sem_ident           ( name, nid, sid )
sem_claim           ( sid, options, time_out )
sem_release         ( sid )
sem_info            ( sid, options, count, tasks_waiting )
```

Queue Operations

```
queue_create        ( name, max_buff, length, options, qid )
```

```
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, msg_length, 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, event, tmid )
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 )
```

```
#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 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_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 ;
```

/*

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 arg1, 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*/
```

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
*/
```

/* 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
```

/* 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
```

/* 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
```

/* 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
```

/* Queue Operations */

```
#ifdef __ANSI__

extern int okqcre( char *name, 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 *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
```

/* Event Operations */

```
#ifdef __ANSI__

extern int okesnd( task_id tid, bit_field event ) ;
extern int okercv( bit_field event, bit_field options, int time_out,
                   bit_field *event_received ) ;

#else

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

#endif


#define event_send        okesnd
#define event_receive     okercv
```

/* Exception operations */

```
#ifdef __ANSI__

extern int okxcat( int bit_number, void new_xsr(bit_field), bit_field
                   new_mode, void (*old_xsr)(bit_field), bit_field
                   *old_mode ) ;
extern int okxrse( task_id tid, bit_field exception ) ;
extern void okxret( void ) ;

#else

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

#endif


#define exception_catch      okxcat
#define exception_raise      okxrse
#define exception_return     okxret
```

/* Clock Operations */

```
#ifdef __ANSI__

extern int okcset( clock_buff *clock ) ;
extern int okcget( clock_buff *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_buff *clock ) ;
extern int oktmea( int ticks, bit_field event, timer_id *tmid ) ;
extern int oktmew( clock_buff *clock, bit_field event, timer_id *tmid );
extern int oktmee( int ticks, bit_field event, timer_id *tid ) ;
extern int oktmca( timer_id *tmid ) ;

#else

extern int oktmwa( ) ;
extern int oktmww( ) ;
extern int oktmea( ) ;
extern int oktmew( ) ;
extern int oktmee( ) ;
extern int oktmca( ) ;

#endif

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

```
/*  Interrupt Operations        */

#ifdef __ANSI__

extern int okient( void ) ;
extern void okiret( void ) ;

#else

extern int okient( ) ;
extern void okiret( ) ;

#endif


#define int_enter   okient
#define int_return  okiret
```

```
/*
```

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. */

```
#define OK                          ???
#define CLOCK_NOT_SET               ???
#define ILLEGAL_USE                 ???
#define INVALID_ARGUMENT            ???
#define INVALID_BIT                 ???
#define INVALID_BUFF                ???
#define INVALID_BUFF_SIZE           ???
#define INVALID_CLOCK               ???
#define INVALID_COUNT               ???
#define INVALID_GRANULARITY         ???
#define INVALID_ID                  ???
#define INVALID_LENGTH              ???
#define INVALID_LOCATION            ???
#define INVALID_NODE                ???
#define INVALID_OPTIONS             ???
#define INVALID_PARAMETER           ???
#define INVALID_PRIORITY            ???
#define INVALID_SEGMENT             ???
#define NAME_NOT_FOUND              ???
#define NODE_NOT_REACHABLE          ???
#define NO_EVENT                    ???
#define NO_MORE_MEMORY              ???
#define OBJECT_DELETED              ???
#define OBJECT_NOT_LOCAL            ???
#define OBJECT_PROTECTED            ???
#define POOL_IN_USE                 ???
#define POOL_NOT_SHARED             ???
#define POOL_OVERLAP                ???
#define QUEUE_DELETED               ???
#define QUEUE_EMPTY                 ???
#define QUEUE_FULL                  ???
#define REGION_IN_USE               ???
#define REGION_OVERLAP              ???
#define SEMAPHORE_DELETED           ???
#define SEMAPHORE_NOT_AVAILABLE     ???
#define SEMAPHORE_OVERFLOW          ???
#define SEMAPHORE_UNDERFLOW         ???
#define TASK_ALREADY_STARTED        ???
#define TASK_ALREADY_SUSPENDED      ???
#define TASK_NOT_STARTED            ???
#define TASK_NOT_SUSPENDED          ???
#define TIME_OUT                    ???
#define TOO_MANY_OBJECTS            ???
#define XSR_NOT_SET                 ???
```

```
/*
```

LITERAL VALUES

This section of the ORKID C language binding contains definitions for all special symbols used as 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. */

```
#define LOCAL_NODE      ???         /* nid */
#define OTHER_NODES     ???
#define ALL_NODES

#define WHO_AM_I        ???         /* name */

#define SELF            ???         /* tid */

#define RUNNING         ???         /* state */
#define READY           ???
#define BLOCKED         ???
#define SUSPENDED       ???

#define CURRENT         ???         /* new_prio */
#define HIGHP           ???         /* new_prio, old_prio */

#define NOXSR           ???         /* new_mode, mode, mask, old_mode */
#define NOTERMINATION   ???
#define NOPREEMPT       ???
#define NOINTERRUPT     ???
#define ALL             ???         /* mask */

#define GLOBAL          ???         /* options */
#define FORCED_DELETE   ???
#define FIFO            ???
#define ANY             ???
#define NOWAIT          ???
#define URGENT          ???
#define ZERO            ???         /* options, mask, modes */

#define FOREVER         ???         /* time_out */

#define NULL_XSR        ???         /* new_xsr, old_xsr */

#endif
```