RTEMS ITRON 3.0 User's Guide

Edition 4.9.1, for RTEMS 4.9.1

12 December 2008

On-Line Applications Research Corporation

On-Line Applications Research Corporation
TEXinfo 2007-12-02.17

The RTEMS Project is hosted at http://www.rtems.com. Any inquiries concerning RTEMS, its related support components, its documentation, or any custom services for RTEMS should be directed to the contacts listed on that site. A current list of RTEMS Support Providers is at http://www.rtems.com/support.html.

Table of Contents

Preface

There needs to be a preface to this manual.

1 Task Manager

1.1 Introduction

The task manager is used to directly control and access the state of tasks. Included among these are functions for creating, deleting, starting and terminating tasks, for releasing the WAIT state of tasks, for enabling/disabling task dispatching, for changing task priority levels, for rotatingtasks on the ready queue, and for accessing task state.

The services provided by the task manager are:

- `cre_tsk` - Create Task
- `del_tsk` - Delete Task
- `sta_tsk` - Start Task
- `ext_tsk` - Exit Issuing Task
- `exd_tsk` - Exit and Delete Issuing Task
- `ter_tsk` - Terminate Other Task
- `dis_dsp` - Disable Dispatch
- `ena_dsp` - Enable Dispatch
- `chg_pri` - Change Task Priority
- `rot_rdq` - Rotate Tasks on the Ready Queue
- `rel_wai` - Release Wait of Other Task
- `get_tid` - Get Task Identifier
- `ref_tsk` - Reference Task Status

1.2 Background

1.2.1 Task Definition

Many definitions of a task have been proposed in computer literature. Unfortunately, none of these definitions encompasses all facets of the concept in a manner which is operating system independent. Several of the more common definitions are provided to enable each user to select a definition which best matches their own experience and understanding of the task concept:

- a "dispatchable" unit.
- an entity to which the processor is allocated.
- an atomic unit of a real-time, multiprocessor system.
- single threads of execution which concurrently compete for resources.
- a sequence of closely related computations which can execute concurrently with other computational sequences.
- From our implementation perspective, a task is the smallest thread of execution which can compete on its own for system resources. A task is manifested by the existence of a task control block (TCB).

1.2.2 Task Manager Task Control Block

The Task Control Block (TCB) is a defined data structure which contains all the information that is pertinent to the execution of a task. During system initialization, implementation reserves a TCB for each task configured. A TCB is allocated upon creation of the task and is returned to the TCB free list upon deletion of the task.

The TCB's elements are modified as a result of system calls made by the application in response to external and internal stimuli. The TCB contains a task's name, ID, current priority, current and starting states, TCB user extension pointer, scheduling control structures, as well as data required by a blocked task.

A task's context is stored in the TCB when a task switch occurs. When the task regains control of the processor, its context is restored from the TCB. When a task is restarted, the initial state of the task is restored from the starting context area in the task's TCB.

1.2.3 T_CTSK Structure

The T_CTSK structure contains detailed information necessary to create the task. Such task attributes, start address, priority and stack size.

```
typedef struct t_ctsk {
  VP    exinf;     /* extended information */
  ATR   tskatr;    /* task attributes */
  FP    task;      /* task start address */
  PRI   itskpri;   /* initial task priority */
  INT   stksz;     /* stack size */
  /* additional implementation dependent information may be included */
} T_CTSK;
```

1.2.4 Task Manager Task States

A task may exist in one of the following five states:

- RUN - Currently scheduled to the CPU
- READY - May be scheduled to the CPU
- Wait - Unable to be scheduled to the CPU
    - (Specific) WAIT - The task is issued a command to wait on a condition
    - SUSPEND - Another task suspended execution of the task
    - WAIT-SUSPEND - Both the WAIT and SUSPEND states apply
- DORMANT - Created task that is not started
- NON-EXISTENT - Uncreated or deleted task

An active task may occupy the RUN, READY, Wait or DORMANT state, otherwise the task is considered NON-EXISTENT. One or more tasks may be active in the system simultaneously. Multiple tasks communicate, synchronize, and compete for system resources with each other via system calls. The multiple tasks appear to execute in parallel, but actually each is dispatched to the CPU for periods of time determined by the scheduling algorithm. The scheduling of a task is based on its current state and priority.

1.2.5 Task Manager Task Priority

A task's priority determines its importance in relation to the other tasks executing on the same processor. Our implementation supports 255 levels of priority ranging from 1 to 255. Tasks of numerically smaller priority values are more important tasks than tasks of numerically larger priority values. For example, a task at priority level 5 is of higher privilege than a task at priority level 10. There is no limit to the number of tasks assigned to the same priority.

Each task has a priority associated with it at all times. The initial value of this priority is assigned at task creation time. The priority of a task may be changed at any subsequent time.

Priorities are used by the scheduler to determine which ready task will be allowed to execute. In general, the higher the logical priority of a task, the more likely it is to receive processor execution time.

1.3 Operations

1.3.1 Task Manager Creating Tasks

The cre_tsk directive creates a task specified by tskid. Specifically, a TCB (Task Control Block) is allocated for the task to be created, and initialized according to accompanying parameter values of itskpri, task, stksz, etc. A stack area is also allocated for the task based on the parameter stksz.

1.3.2 Task Manager Starting and Restarting Tasks

The sta_tsk directive starts the task specified by tskid. Specifically, it changes the state of the task specified by tskid from DORMANT into RUN/READY. This enables the task to compete, based on its current priority, for the processor and other system resources. Any actions, such as suspension or change of priority, performed on a task prior to starting it are nullified when the task is started.

Stacd can be used to specify parameters to be passed to the task when it is started. This parameter can be read by the task being started, and may be used for transmitting simple messages.

The task priority on starting the task is given by the initial task priority parameter (itskpri) specified when the task was created.

Start request is not queued in this this system call. In other words, if this system call is issued when the target task is not in DORMANT state, the system call will be ignored, and an E_OBJ error returned to the issuing task.

If cre_tsk [level EN] is not implemented on a system, tasks are created statically when the system is started. Parameters required for creating a task, such as task starting address (task) and initial task priority (itskpri) are also specified statically at system startup.

1.3.3 Task Manager Suspending and Resuming Tasks

The sus_tsk directive suspends the execution of the task specified by tskid by putting it into SUSPEND state. SUSPEND state is released by issuing the rsm_tsk or frsm_tsk system call.

If the task specified to sus_tsk is already in WAIT state, it will be put in the combined WAIT-SUSPEND state by the execution of sus_tsk. If wait conditions for the task are later fulfilled, it will enter SUSPEND state. If rsm_tsk is issued on the task, it will return to the WAIT state before the suspension.

Both rsm_tsk and fsm_tsk system calls release SUSPEND state of the task specified by tskid. Specifically, they cause SUSPEND state to be released and the execution of the specified task to resume when the task has been suspended by the prior execution of sus_tsk.

If the specified task is in WAIT-SUSPEND state, the execution of rsm_tsk only releases the SUSPEND state, and the task will become WAIT state.

1.3.4 Task Manager Changing Task Priority

The chg_pri system call changes the current priority of the task specified by tskid to the value specified by tskpri.

A task may specify itself by specifying tskid = TSK_SELF = 0. Note, however, that an E_ID error will result if tskid = TSK_SELF = 0 is specified to a system call issued from a task-independent portion.

The priority set by this system call remains in effect until the task exits. Once a task enters DORMANT state, its priority prior to exiting is lost. When a task which enters DORMANT state restarts, the initial task priority (itskpri) specified at task creation or at system startup will be used.

1.3.5 Task Manager Task Deletion

The del_tsk system call deletes the task specified by tskid. Specifically, it changes the state of the task specified by tskid from DORMANT into NON-EXISTENT (a virtual state not existing on the system), and then clears the TCB and releases stack. An E_OBJ error results if this system call is used on a task which is not DORMANT.

After deletion, another task having the same ID number can be created.

The exd_tsk system call causes the issuing task to exit and then delete itself.

When a task exits, that task does not automatically release all the resources (memory blocks, semaphores, etc.) which it had secured prior to the call. It is the user's responsibility to see to it that all resources are released beforehand.

1.4 System Calls

This section details the task manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

1.4.1 cre_tsk - Create Task

CALLING SEQUENCE:

```
ER cre_tsk(
  ID tskid,
  T_CTSK *pk_ctsk
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_NOMEM` - Insufficient memory (Memory for control block and/or user stack cannot be allocated)

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_RSATR` - Reserved attribute (tskatr was invalid or could not be used)

`E_OBJ` - Invalid object state (a task of the same ID already exists)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (pk_ctsk, task, itskpri and/or stksz is invalid)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for exinf, tskatr, task, itskpri and/or stksz)

DESCRIPTION:

This system call creates the task specified by tskid. Specifically, a TCB (Task Control Block) is allocated for the task to be created, and initialized according to accompanying parameter values of itskpri, task, tksz, etc. A stack area is also allocated for the task based on the parameter stksz.

NOTES:

User tasks have positive ID numbers, while system tasks have negative ID numbers. User tasks cannot access system objects (objects having negative ID numbers).

The new task created by this system call will be put in DORMANT state.

Extended information (exinf) has been added. This allows the user to include additional information about task attributes. If a larger region is desired for including user information, the user should allocate memory area and set the address of the memory packet to exinf.

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

1.4.2 del_tsk - Delete Task

CALLING SEQUENCE:

```
ER del_tsk(
  ID tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_OBJ` - Invalid object state (the target task is not in DORMANT state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This system call deletes the task specified by tskid. Specifically, it changes the state of the task specified by tskid from DORMANT into NON-EXISTENT (a virtual state not existing on the system), and then clears the TCB and releases stack. An E_OBJ error results if this system call is used on a task which is not DORMANT.

After deletion, another task having the same ID number can be created.

NOTES:

A task cannot delete itself by this system call. An E_OBJ error will result if a task specifies itself, since such a task cannot be DORMANT. Use the exd_tsk system call rather than this one when a task needs to delete itself.

1.4.3 sta_tsk - Start Task

CALLING SEQUENCE:

```
ER sta_tsk(
  ID tskid,
  INT stacd
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_OBJ` - Invalid object state (the target task is not in DORMANT state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for stacd)

DESCRIPTION:

This system call starts the task specified by tskid. Specifically, it changes the state of the task specified by tskid from DORMANT into RUN/READY.

Stacd can be used to specify parameters to be passed to the task when it is started. This parameter can be read by the task being started, and may be used for transmitting simple messages.

The task priority on starting the task is given by the initial task priority parameter (itskpri) specified when the task was created.

Start request is not queued in this this system call. In other words, if this system call is issued when the target task is not in DORMANT state, the system call will be ignored, and an E_OBJ error returned to the issuing task.

If cre_tsk [level EN] is not implemented on a system, tasks are created statically when the system is started. Parameters required for creating a task, such as task starting address (task) and initial task priority (itskpri) are also specified statically at system startup.

NOTES:

1.4.4 ext_tsk - Exit Issuing Task

CALLING SEQUENCE:

```
void ext_tsk(void);
```

STATUS CODES:

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

* System call may detect this error. The error is not returned to the context issuing the system call. Error codes therefore cannot be returned directly as a return parameter of the system call. The behavior on error detection is implementation dependent.

DESCRIPTION:

This system call causes the issuing task to exit, changing the state of the task into the DORMANT state.

NOTES:

When a task exits due to ext_tsk, that task does not automatically release all the resources (memory blocks, semaphores, etc.) which it had obtained prior to the system call. It is the user's responsibility that all resources are released beforehand.

Ext_tsk is a system call which does not return to the issuing context. Accordingly, even if an error code is returned on detection of some error, it is normal for tasks making this system call not to perform any error checking, and it is in fact possible that a program could run out of control. For this reason, even if an error is detected upon issuing this system call, the error is not returned to the task which issued the system call. If information on detected errors is required it should be left in a messagebuffer used as an error log.

In principle, information concerning a task recorded in the TCB, such as task priority, is reset whenever a task is placed in DORMANT state. For example, its task priority after being restarted would be reset to the initial task priority (itskpri) specified by cre_tsk when it was first created, even if a task's priority was changed using chg_pri, then that task exits using ext_tsk, but later started by sta_tsk. Task priority does not return to what it was when ext_tsk was executed.

1.4.5 exd_tsk - Exit and Delete Issuing Task

CALLING SEQUENCE:

```
void exd_tsk(void);
```

STATUS CODES:

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

* System call may detect the following error. The error is not returned to the context issuing the system call even. Error codes therefore cannot be returned directly as a return parameter of the system call. The behavior on error detection is implementation dependent.

DESCRIPTION:

This system call causes the issuing task to exit and then delete itself. In other words the state of the issuing task changes into the NON-EXISTENT (a virtual state not existing on the system).

NOTES:

When a task exits with exd_tsk, that task does not automatically release all the resources (memory blocks, semaphores, etc.) which it had secured prior to the call. It is the user's responsibility to see to it that all resources are released beforehand.

Exd_tsk is a system call which does not return any parameters to the original issuing context. Accordingly, even if an error code is returned on detection of some error, it is normal for tasks making this system call not to perform any error checking, and it is in fact possible that a program could run out of control. For this reason, even if an error is detected upon making this system call, it is supposed that the error is not returned to the task which issued the system call. If information on detected errors is required it should be left in a messagebuffer used as an error log.

1.4.6 ter_tsk - Terminate Other Task

CALLING SEQUENCE:

```
ER ter_tsk(
  ID tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_OBJ` - Invalid object state (the target task is already in DORMANT state or a task invalidly specified itself)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This system call forcibly terminates the task specified by tskid. That is, it changes the state of the task specified by tskid into DORMANT.

Even if the target task is in wait state (including SUSPEND state), its wait state will be released and then it will be terminated. If the target task is on a queue of some sort (such as waiting for a semaphore), it will be removed from that queue by ter_tsk.

A task cannot specify the issuing task in this system call. An E_OBJ error will result if a task specifies itself.

There is an intermediate state waiting for the response (TR packet or TA packet) from the target node after executing the system call to access the other node and making a request (sending a TP packet) to the node. This state is called the "connection function response wait (TTW_NOD)" state. The ter_tsk system call may specify tasks which are in the connection function response wait state. Tasks which are waiting for objects (such as a semaphore) on another node may also be specified to this system call. In such cases, ter_tsk will halt any system calls accessing other nodes which have been issued by the task to be terminated.

NOTES:

When a task is terminated by ter_tsk, that task does not automatically release all the resources (memory blocks, semaphores, etc.) which it had obtained prior to the call. It is the user's responsibility to see to it that all resources are released beforehand.

In principle, information concerning a task recorded in the TCB, such as task priority, is reset whenever a task is placed in DORMANT state. For example, its task priority after being restarted would be reset to the initial task priority (itskpri) specified by cre_tsk when it was first created, even if a task's priority was changed using chg_pri, then that task is terminated by ter_tsk, but later started by sta_tsk. Task priority does not return to what it was when ter_tsk was executed.

1.4.7 dis_dsp - Disable Dispatch

CALLING SEQUENCE:

```
ER dis_dsp(void);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_CTX` - Context error (issued from task-independent portions or issued after execution of loc_cpu)

DESCRIPTION:

This system call disables task dispatching. Dispatching will remain disabled after this call is issued until a subsequent call to ena_dsp is issued. The status of the issuing task will not be allowed to be changed to READY from the RUN. It cannot be changed into WAIT, either. However, since external interrupt is not disabled, interrupt handlers are allowed to run even when dispatching has been disabled. While an executing task may be preempted by an interrupt handler with dispatching disabled, there is no possibility that it will be preempted by another task.

The following operations occur during the time dispatching is disabled.

- Even in a situation where normally a task issuing dis_dsp should be preempted by a system call issued by an interrupt handler or by the task issuing dis_dsp, the task that should normally be executed is not dispatched. Instead, dispatching of this task is delayed until dispatch disabled state is cleared by ena_dsp.
- If an interrupt handler invoked during dispatch disabled state issues sus_tsk for a running task (one that executed dis_dsp) to put it in SUSPEND state, or ter_tsk to put it in DORMANT state, the task transition is delayed until dispatch disabled state is cleared.
- An E_CTX error will result if the task which has executed dis_dsp issues any system calls (such as slp_tsk or wai_sem) capable of putting an issuing task into WAIT state.
- An EN_CTXID error will result if a task which has executed dis_dsp attempts to operate on objects on another node (that is, if the ID parameter of the system call issued refers to an object on another node).
- TSS_DDSP will be returned as sysstat if system status is referenced using ref_sys.

No error will result if a task already in dispatch disable state issues dis_dsp. It only keeps dispatch disabled state. No matter how many times dis_dsp has been issued, a single ena_dsp enables dispatching again. It is therefore for the user to determine what to do with nested pairs of dis_dsp and ena_dsp.

An E_CTX error will result if dis_dsp is issued when both interrupt and dispatching are disabled with loc_cpu. (For details, see the description of loc_cpu.)

NOTES:

A running task cannot enter DORMANT or NON-EXISTENT state while dispatching is disabled. An E_CTX error will result if an running task issues either ext_tsk or exd_tsk while interrupt and dispatching are disabled. Note however that since both ext_tsk and exd_tsk are system calls which do not return to their original contexts, error notification using return parameters of these system calls is not possible. If information on detected errors is required it should be left in a messagebuffer used as an error log.

Only if the system is not a multiprocessor configuration, system can take advantage of the dispatch disabled state for exclusive inter-task control.

1.4.8 ena_dsp - Enable Dispatch

CALLING SEQUENCE:

```
ER ena_dsp(void);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_CTX` - Context error (issued from task-independent portions or issued after execution of loc_cpu)

DESCRIPTION:

This system call enables task dispatching, that is, it finishes dispatch disabled state caused by the execution of dis_dsp.

No error will result if a task which is not in dispatch disabled state issues ena_dsp. In this case, dispatching will just remain enabled.

An E_CTX error will result if ena_dsp is issued when both interrupt and dispatching are disabled with loc_cpu. (For details, see the description of loc_cpu.)

NOTES:

1.4.9 chg_pri - Change Task Priority

CALLING SEQUENCE:

```
ER chg_pri(
  ID tskid,
  PRI tskpri
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (the value of tskpri is invalid or may not be used)

`E_OBJ` - Invalid object state (the target task is in DORMANT state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` = Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for tskpri)

DESCRIPTION:

This system call changes the current priority of the task specified by tskid to the value specified by tskpri.

Under uITRON 3.0 specification, at least any value of 1 through 8 can be specified as task priority. The smaller the value, the higher the priority. Priority levels -4 through 0 are reserved, and they may not be used. Priority levels outside this range (including negative values) may also be specified depending on the implementation; this is considered an extended function [level X] for which compatibility and connectivity are not guaranteed. In general, negative priority levels are reserved for use by the system.

A task may specify itself by specifying tskid = TSK_SELF = 0. Note, however, that an E_ID error will result if tskid = TSK_SELF = 0 is specified to a system call issued from a task-independent portion. The priority set by this system call remains in effect until the task exits. Once a task enters DORMANT state, its priority prior to exiting is lost. When a task which enters DORMANT state restarts, the initial task priority (itskpri) specified at task creation or at system startup will be used.

If the target task is linked to ready queue or any other queue, this system call may result in the re-ordering of the queues. If chg_pri is executed on a task waiting on the ready queue (including tasks in RUN state) or other priority-based queue, the target task will be moved to the end of the part of the queue for the associated priority. If the priority specified is the same as the current priority, the task will still be moved behind other tasks of the same priority. It is therefore possible for a task to relinquish its execution privileges using chg_pri on itself by specifying its current priority.

NOTES:

Depending on the implementation, specifying tskpri = TPRI_INI = 0 may cause a task's priority to be reset to the initial task priority (itskpri) which was defined when it was first created or when the system started. This feature is used in some implementations in order to reset the task priority to its original value after setting it to a higher value for indivisible processing. This feature is an extended function [level X] for which compatibility and connectivity are not guaranteed.

1.4.10 rot_rdq - Rotate Tasks on the Ready Queue

CALLING SEQUENCE:

```
ER rot_rdq(
  PRI tskpri
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_PAR` - Parameter error (the value of tskpri is invalid)

DESCRIPTION:

This system call rotates tasks on the ready queue associated with the priority level specified by tskpri. Specifically, the task at the head of the ready queue of the priority level in question is moved to the end of the ready queue, thus switching the execution of tasks having the same priority. Round robin scheduling may be implemented by periodically issuing this system call in a given period of time.

When rot_rdq is issued by task portions with tskpri = TPRI_RUN = 0, the ready queue with the priority level of the issuing task is rotated.

When TPRI_RUN or a task's own priority level are specified for tskpri to rot_rdq, the task issuing the system call will be placed on the end of its ready queue. In other words, task can issue rot_rdq to relinquishing its execution privileges. The concept of "ready queue" envisioned in the description of this system call is one which includes the task in RUN state.

This system call does nothing if there are no tasks on the ready queue of the specified priority. No error will result.

This system call cannot rotate ready queues on other nodes.

NOTES:

Depending on the implementation, it may be possible to issue rot_rdq(tskpri = TPRI_RUN) from task-independent portions, such as a cyclic handler. In this case the ready queue including the running task, or the ready queue including the highest priority task, is rotated. Normally these two are the same, but not always, as when task dispatching is delayed. In that case it is implementation dependent whether to rotate the ready queue including the running task or the ready queue including the highest priority task. Note that this is an extended function [Level X] for which compatibility and connectivity are not guaranteed.

1.4.11 rel_wai - Release Wait of Other Task

CALLING SEQUENCE:

```
ER rel_wai(
  ID tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_OBJ` - Invalid object state (the target task is not in WAIT state (including when it is in DORMANT state or when the issuing task specifies itself))

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This system call forcibly releases WAIT state (not including SUSPEND state) of the task specified by tskid.

An E_RLWAI error is returned to the task whose WAIT state has been released using rel_wai.

Wait release requests by rel_wai are not queued. In other words, if the task specified by tskid is already in WAIT state, the WAIT state is released, otherwise an E_OBJ error will be returned to the issuer. An E_OBJ error will also result when a task specifies itself to this system call.

Rel_wai does not release SUSPEND state. If rel_wai is issued on a task in WAIT-SUSPEND state, WAIT will be released but SUSPEND will continue for that task. When SUSPEND should also be released, the frsm_tsk system call must be issued separately.

NOTES:

A function similar to timeout can be implemented using an alarm handler which issues this system call on tasks specified time after they have entered WAIT state.

Rel_wai and wup_tsk differ in the following points.

- Wup_tsk can only release the WAIT state by slp_tsk or tslp_tsk, while rel_wai can release WAIT states caused by these and other calls (including wai_flg, wai_sem, rcv_msg, get_blk, etc.).

- As seen from the target task, releasing WAIT state with wup_tsk results in a normal completion (E_OK), whereas releasing WAIT state with rel_wai results in an error (E_RLWAI).
- When wup_tsk is used, a request is queued even if neither slp_tsk nor tslp_tsk have been executed on the target task yet. When rel_wai is used to the task which is not in WAIT state, an E_OBJ error will result.

1.4.12 get_tid - Get Task Identifier

CALLING SEQUENCE:

```
ER get_tid(
  ID *p_tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

DESCRIPTION:

This system call gets the ID of the issuing task.

If this system call is issued from a task-independent portion, tskid will be FALSE=0.

NOTES:

1.4.13 ref_tsk - Reference Task Status

CALLING SEQUENCE:

```
ER ref_tsk(
  T_RTSK *pk_rtsk,
  ID tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (the packet address for return parameters cannot be used)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_RPAR` - A value outside the range supported by the requesting node and/or transmission packet format was returned as a return parameter (a value outside supported range was returned for exinf, tskpri and/or tskstat)

DESCRIPTION:

This system call refers to the state of the task specified by tskid, and returns its current priority (tskpri), its task state (tskstat), and its extended information (exinf).

Tskstat may take the following values.

tskstat:

- TTS_RUN H'0...01 RUN state (currently running)
- TTS_RDY H'0...02 READY state (ready to run)
- TTS_WAI H'0...04 WAIT state (waiting for something)
- TTS_SUS H'0...08 SUSPEND state (forcibly made to wait)
- TTS_WAS H'0...0c WAIT-SUSPEND state
- TTS_DMT H'0...10 DORMANT state

Since these task states are expressed by bit correspondences they are convenient when looking for OR conditions (such as whether a task is in RUN or READY state). TTS_WAS is a combination of both TTS_SUS and TTS_WAI, TTS_SUS does not combine with any of the other states (TTS_RUN, TTS_RDY or TTS_DMT).

A task may specify itself by specifying tskid = TSK_SELF = 0. Note, however, that an E_ID error will result if tskid = TSK_SELF = 0 is specified when this system call is issued from a task-independent portion.

An E_NOEXS error will result if the task specified to ref_tsk does not exist.

Tskstat will be TTS_RUN if ref_tsk is executed specifying a task which has been interrupted by an interrupt handler.

NOTES:

The values of TTS_RUN, TTS_RDY, TTS_WAI, etc. as return values for tskstat are not necessarily the same value to be entered in the TCB. The way in which task state is represented in the TCB is implementation dependent. When ref_tsk is executed, the internal representation of task state may simply be converted to the standard values TTS_RUN, TTS_RDY, TTS_WAI, etc.

Depending on the implementation, the following additional information can also be referenced in addition to exinf, tskpri and tskstat.

- tskwait Reason for wait
- wid Wait object ID
- wupcnt Number of queued wakeup requests
- suscnt Number of nested SUSPEND requests
- tskatr Task attributes
- task Task starting address
- itskpri Initial task priority
- stksz Stack size

2 Task-Dependent Synchronization Manager

2.1 Introduction

The task-dependent synchronization manager is designed to utilize those synchronization functions already supported by tasks. This includes functions that suspend tasks for a while and associated functions that release SUSPEND state, and synchronization functions which make tasks wait and wake them up.

The services provided by the task-dependent synchronization manager are:

- `sus_tsk` - Suspend Other Task
- `rsm_tsk` - Resume Suspended Task
- `frsm_tsk` - Forcibly Resume Suspended Task
- `slp_tsk` - Sleep Task
- `tslp_tsk` - Sleep Task with Timeout
- `wup_tsk` - Wakeup Other Task
- `can_wup` - Cancel Wakeup Request

2.2 Operations

2.2.1 Suspend Other Task

This call stops the execution of a task by putting it into a SUSPEND state. This call is not able to specify itself, since this would end the flow of execution altogether. If the task is already in a WAIT state, then SUSPEND is added to become WAIT-SUSPEND. These modes are turned on and off separately, without affecting one another. Furthermore, SUSPEND states can be nested, and tasks in a SUSPEND state are allocated resources as normal.

2.2.2 Resume Suspended Task

This operation restarts the execution of a task that was previously stopped by the SUSPEND OTHER TASK call. Obviously, a task cannot specify itself using this call. Since SUSPEND states can be nested, one call to RESUME releases only one SUSPEND. Thus, it takes as many RESUMES as SUSPENDS to return the task to execution.

2.2.3 Forcibly Resume Suspended Task

This call has the same functionality as the previously mentioned Resume Suspended Task with one exception. This call releases all nested SUSPENDS at once, which guarantees the task will return to execution.

2.2.4 Sleep Task

The Sleep Task operation causes the specified task to sleep until a Wakeup Task function is called. This puts the task in a WAIT state. WAIT states can not be nested, but can be combined with SUSPEND states as mentioned earlier.

2.2.5 Sleep Task with Timeout

This function is identical to the Sleep Task function with an added timeout attribute. If the timeout mark is reached before a Wakeup call is recieved, an error is generated.

2.2.6 Wakeup Other Task

The Wakeup Other Task call is used to release the WAIT state of a task. These calls can be previously queued using the wupcnt value so that when the matching Sleep Task is executed, there will be no delay.

2.2.7 Cancel Wakeup Request

This function call resets the value of wupcnt to zero, thereby canceling all associated wakeup requests. A call to self is acceptable for this operation, and may even be useful for monitoring certain situations.

2.3 System Calls

This section details the task-dependent synchronization manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

2.3.1 sus_tsk - Suspend Other Task

CALLING SEQUENCE:

```
ER sus_tsk(
  ID tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_OBJ` - Invalid object state (the specified task is in DORMANT state or the issuing task specified itself)

`E_QOVR` - Queuing or nesting overflow (the number of nesting levels given by suscnt went over the maximum allowed)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This system call suspends the execution of the task specified by tskid by putting it into SUSPEND state.

SUSPEND state is released by issuing the rsm_tsk or frsm_tsk system call. If the task specified to sus_tsk is already in WAIT state, it will be put in the combined WAIT-SUSPEND state by the execution of sus_tsk. If wait conditions for the task are later fulfilled, it will enter SUSPEND state. If rsm_tsk is issued on the task, it will return to the WAIT state before the suspension.

Since SUSPEND state indicates the suspension of execution by a system call issued from another task, a task may not specify itself to this system call. An E_OBJ error will result if a task specifies itself.

If more than one sus_tsk call is issued to a task, that task will be put in multiple SUSPEND states. This is called suspend request nesting. When this is done, rsm_tsk must be issued the same number of times which sus_tsk was issued (suscnt) in order to return the task to its original state before the suspension. This means it is possible to nest the pairs of sus_tsk and rsm_tsk.

The maximum number of times suspend requests may be nested, and even whether or not suspend request nesting (the ability to issue sus_tsk on the same task more than once) is even allowed, is implementation dependent. Suspend request nesting is considered an extended function [level X] for which compatibility and connectivity are not guaranteed.

An E_QOVR error will result if sus_tsk is issued more than once on the same task on a system which does not support suspend request nesting or if it is issued more than the maximum number of times allowed.

NOTES:

A task which is suspended in addition to waiting for resources (such as waiting for a semaphore) can be allocated resources (such as semaphore counts) based on the same conditions as tasks which are not suspended. Even when suspended, the allocation of resources is not delayed in any way. Conditions concerning resource allocation and release of the wait state remain unchanged. In other words, SUSPEND state is completely independent of other processing and task states. If it is desirable to delay the allocation of resources to a task which is suspended, the user should use chg_pri in conjunction with sus_tsk and rsm_tsk.

2.3.2 rsm_tsk - Resume Suspended Task

CALLING SEQUENCE:

```
ER rsm_tsk(
  ID tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_OBJ` - Invalid object state (the target task is not in SUSPEND state (including when it is DORMANT or when the issuing task specifies itself))

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This system call releases SUSPEND state of the task specified by tskid. Specifically, it causes SUSPEND state to be released and the execution of the specified task to resume when the task has been suspended by the prior execution of sus_tsk. If the specified task is in WAIT-SUSPEND state, the execution of rsm_tsk only releases the SUSPEND state, and the task will become WAIT state.

A task cannot specify itself to this system call. An E_OBJ error will result if a task specifies itself.

Rsm_tsk only releases one suspend request from the suspend request nest (suscnt). Accordingly, if more than one sus_tsk has been issued on the task in question (suscnt >= 2), that task will remain suspended even after the execution of rsm_tsk is completed.

NOTES:

It is implementation dependent which location in the ready queue a task returns to after the task which has been suspended from RUN or READY state is resumed by rsm_tsk.

2.3.3 frsm_tsk - Forcibly Resume Suspended Task

CALLING SEQUENCE:

```
ER ercd =frsm_tsk(
  ID tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_OBJ` - Invalid object state (the target task is not in SUSPEND state (including when it is DORMANT or when the issuing task specifies itself))

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This system call releases SUSPEND state of the task specified by tskid. Specifically, it causes SUSPEND state to be released and the execution of the specified task to resume when the task has been suspended by the prior execution of sus_tsk. If the specified task is in WAIT-SUSPEND state, the execution of rsm_tsk only releases the SUSPEND state, and the task will become WAIT state.

A task cannot specify itself to this system call. An E_OBJ error will result if a task specifies itself.

Frsm_tsk will clear all suspend requests (suscnt = 0) even if more than one sus_tsk has been issued (suscnt >= 2) on the same task. In other words, SUSPEND state is guaranteed to be released, and execution will resume unless the task in question had been in combined WAIT-SUSPEND state.

NOTES:

It is implementation dependent which location in the ready queue a task returns to after the task which has been suspended from RUN or READY state is resumed by frsm_tsk.

2.3.4 slp_tsk - Sleep Task Sleep Task with Timeout

CALLING SEQUENCE:

```
ER slp_tsk( void );
```

STATUS CODES:

`E_OK` - Normal Completion

`E_PAR` - Parameter error (a timeout value -2 or less was specified)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

DESCRIPTION:

This system call puts the issuing task (which was in RUN state) into WAIT state, causing the issuing task to sleep until wup_tsk is invoked.

NOTES:

Since the slp_tsk system call causes the issuing task to enter WAIT state, slp_tsk calls may not be nested. It is possible, however, for another task to execute a sus_tsk on a task which has put itself in WAIT state using slp_tsk. If this happens, the task will enter the combined WAIT-SUSPEND state.

No polling function for slp_tsk is provided. A similar function can be implemented if necessary using can_wup.

2.3.5 tslp_tsk - Sleep Task with Timeout

CALLING SEQUENCE:

```
ER ercd =tslp_tsk(
  TMO tmout
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_PAR` - Parameter error (a timeout value -2 or less was specified)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

DESCRIPTION:

The tslp_tsk system call is the same as slp_tsk but with an additional timeout feature. If a wup_tsk is issued before the period of time specified by tmout elapses, tslp_tsk will complete normally. An E_TMOUT error will result if no wup_tsk is issued before the time specified by tmout expires. Specifying tmout = TMO_FEVR = -1 can be used to set the timeout period to forever (no timeout). In this case, tslp_tsk will function exactly the same as slp_tsk causing the issuing task to wait forever for wup_tsk to be issued.

NOTES:

Since the tslp_tsk system call causes the issuing task to enter WAIT state, tslp_tsk calls may not be nested. It is possible, however, for another task to execute a sus_tsk on a task which has put itself in WAIT state using tslp_tsk. If this happens, the task will enter the combined WAIT-SUSPEND state.

If you simply wish to delay a task (make it wait for a while), use dly_tsk rather than tslp_tsk.

2.3.6 wup_tsk - Wakeup Other Task

CALLING SEQUENCE:

```
ER wup_tsk(
  ID tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_OBJ` - Invalid object state (the specified task is in DORMANT state or the issuing task specified itself)

`E_QOVR` - Queuing or nesting overflow (wakeup request queuing count will exceed the maximum value allowed for wupcnt)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This system call releases the WAIT state of the task specified by tskid caused by the execution of slp_tsk or tslp_tsk.

A task cannot specify itself in this system call. An E_OBJ error will result if a task specifies itself.

If the specified task is not in the WAIT state caused by a slp_tsk or tslp_tsk, the wakeup request based on the wup_tsk call will be queued. In other words, a record will be kept that a wup_tsk has been issued for the specified task and no WAIT state will result even if slp_tsk or tslp_tsk is executed by the task later. This is called queuing for wakeup request.

NOTES:

Wakeup requests are queued as follows. A wakeup request queuing count (wupcnt) is kept in the TCB for each task. Initially (when sta_tsk is executed) the value of wupcnt is 0. Executing wup_tsk on a task which is not waiting for a wakeup increments the wakeup request queuing count by one for the specified task. If slp_tsk or tslp_tsk is executed on that task, its wakeup request queuing count will be decremented by one. If the task with wakeup request queuing count = 0 executes slp_tsk or tslp_tsk, that task will be put in WAIT state rather than decrementing the wakeup request queuing count.

It is always possible to queue at least one wup_tsk (wupcnt = 1); the maximum allowable number for the wakeup request queuing count (wupcnt) is implementation dependent, and may be any number higher than or equal to one. In other words, while the first wup_tsk issued to a task which is not waiting for a wakeup will not result in an error, it is implementation dependent whether or not any further wup_tsk calls on the same task will result in an error. The ability to queue more than one wakeup request is considered an extended function [level X] for which compatibility and connectivity are not guaranteed.

An E_QOVR error will result if wup_tsk is issued more than the maximum value allowed for the wakeup request queuing count (wupcnt).

2.3.7 can_wup - Cancel Wakeup Request

CALLING SEQUENCE:

```
ER can_wup(
  INT *p_wupcnt,
  ID tskid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID Number (tskid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the task specified by tskid does not exist)

`E_OACV` - Object access violation (A tskid less than -4 was specified from a user task. This is implementation dependent.)

`E_OBJ` - Invalid object state (the target task is in DORMANT state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_RPAR` - A value outside the range supported by the issuing node and/or transmission packet format was returned as a return parameter (a value outside supported range was returned for wupcnt)

DESCRIPTION:

This system call returns the wakeup request queuing count (wupcnt) for the task specified by tskid while canceling all associated wakeup requests. Specifically, it resets the wakeup request queuing count (wupcnt) to 0.

A task may specify itself by specifying tskid = TSK_SELF = 0. Note, however, that an E_ID error will result if tskid = TSK_SELF = 0 is specified when this system call is issued from a task-independent portion.

NOTES:

An EN_RPAR error will result if the number of bits used on the target node is larger than that used on the requesting node, and if a value not supported by the requesting node is returned for wupcnt.

This system call can be used to determine whether or not processing has ended within a certain period when a task should periodically waken up by wup_tsk and do some processing. In other words, if a task monitoring the progress of its processing issues can_wup before issuing a slp_tsk after finishing processing associated with a previous wakeup request, and if wupcnt, one of can_wup's return parameters, is equal to or greater than one, it indicates that the processing for the previous wakeup request does not complete within a required time. This allows the monitoring task to take actions against processing delays.

3 Semaphore Manager

3.1 Introduction

The semaphore manager provides functions to allocate, delete, and control counting semaphores. This manager is based on the ITRON 3.0 standard.

The services provided by the semaphore manager are:

- `cre_sem` - Create Semaphore
- `del_sem` - Delete Semaphore
- `sig_sem` - Signal Semaphore
- `wai_sem` - Wait on Semaphore
- `preq_sem` - Poll and Request Semaphore
- `twai_sem` - Wait on Semaphore with Timeout
- `ref_sem` - Reference Semaphore Status

3.2 Background

3.2.1 Theory

Semaphores are used for synchronization and mutual exclusion by indicating the availability and number of resources. The task (the task which is returning resources) notifying other tasks of an event increases the number of resources held by the semaphore by one. The task (the task which will obtain resources) waiting for the event decreases the number of resources held by the semaphore by one. If the number of resources held by a semaphore is insufficient (namely 0), the task requiring resources will wait until the next time resources are returned to the semaphore. If there is more than one task waiting for a semaphore, the tasks will be placed in the queue.

3.2.2 T_CSEM Structure

The T_CSEM structure is used to define the characteristics of a semaphore and passed an as argument to the `cre_sem` service. The structure is defined as follows:

```
/*
 *  Create Semaphore (cre_sem) Structure
 */

typedef struct t_csem {
  VP    exinf;    /* extended information */
  ATR   sematr;   /* semaphore attributes */
  /* Following is the extended function for [level X]. */
  INT   isemcnt;   /* initial semaphore count */
  INT   maxsem;    /* maximum semaphore count */
  /* additional implementation dependent information may be included */
} T_CSEM;

/*
 *  sematr - Semaphore Attribute Values
 */

#define TA_TFIFO   0x00   /* waiting tasks are handled by FIFO */
#define TA_TPRI    0x01   /* waiting tasks are handled by priority */
```

where the meaning of each field is:

exinf is for any extended information that the implementation may define. This implementation does not use this field.

sematr is the attributes for this semaphore. The only attributed which can be specified is whether tasks wait in FIFO (`TA_TFIFO`) or priority (`TA_TPRI`) order.

isemcnt is the initial count of the semaphore.

maxsem is the maximum count the semaphore may have. It places an upper limit on the value taken by the semaphore.

3.2.3 Building a Semaphore Attribute Set

In general, an attribute set is built by a bitwise OR of the desired attribute components. The following table lists the set of valid semaphore attributes:

- `TA_TFIFO` - tasks wait by FIFO
- `TA_TPRI` - tasks wait by priority

Attribute values are specifically designed to be mutually exclusive, therefore bitwise OR and addition operations are equivalent as long as each attribute appears exactly once in the component list.

3.2.4 T_RSEM Structure

The T_RSEM structure is filled in by the `ref_sem` service with status and state information on a semaphore. The structure is defined as follows:

```
/*
 *  Reference Semaphore (ref_sem) Structure
 */

typedef struct t_rsem {
  VP      exinf;    /* extended information */
  BOOL_ID wtsk;     /* indicates whether there is a waiting task */
  INT     semcnt;   /* current semaphore count */
  /* additional implementation dependent information may be included */
} T_RSEM;
```

exinf is for any extended information that the implementation may define. This implementation does not use this field.

wtsk is TRUE when there is one or more task waiting on the semaphore. It is FALSE if no tasks are currently waiting. The meaning of this field is allowed to vary between ITRON implementations. It may have the ID of a waiting task, the number of tasks waiting, or a boolean indication that one or more tasks are waiting.

semcnt is the current semaphore count.

The information in this table is very volatile and should be used with caution in an application.

3.3 Operations

3.3.1 Using as a Binary Semaphore

Creating a semaphore with a limit on the count of 1 effectively restricts the semaphore to being a binary semaphore. When the binary semaphore is available, the count is 1. When the binary semaphore is unavailable, the count is 0.

Since this does not result in a true binary semaphore, advanced binary features like the Priority Inheritance and Priority Ceiling Protocols are not available.

3.4 System Calls

This section details the semaphore manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

3.4.1 cre_sem - Create Semaphore

CALLING SEQUENCE:

```
ER cre_sem(
  ID semid,
  T_CSEM *pk_csem
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (semid was invalid or could not be used)

`E_NOMEM` - Insufficient memory (Memory for control block cannot be allocated)

`E_OACV` - Object access violation (A semid less than -4 was specified from a user task.)

`E_RSATR` - Reserved attribute (sematr was invalid or could not be used)

`E_OBJ` - Invalid object state (a semaphore of the same ID already exists)

`E_PAR` - Parameter error (pk_csem is invalid and/or isemcnt or maxsem is negative or invalid)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task- independent portion

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for exinf, sematr, isemcnt and/or maxsem)

DESCRIPTION:

This routine creates a semaphore that resides on the local node. The semaphore is initialized based on the attributes specified in the `pk_csem` structure. The initial and maximum counts of the semaphore are set based on the `isemcnt` and `maxsem` fields in this structure.

Specifying `TA_TPRI` in the `sematr` field of the semaphore attributes structure causes tasks waiting for a semaphore to be serviced according to task priority. When `TA_TFIFO` is selected, tasks are serviced in First In-First Out order.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

All memory is preallocated for RTEMS ITRON objects. Thus, no dynamic memory allocation is performed by `cre_sem` and the `E_NOMEM` error can not be returned.

This directive will not cause the running task to be preempted.

The following semaphore attribute constants are defined by RTEMS:

- `TA_TFIFO` - tasks wait by FIFO
- `TA_TPRI` - tasks wait by priority

3.4.2 del_sem - Delete Semaphore

CALLING SEQUENCE:

```
ER del_sem(
  ID semid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (semid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the semaphore specified by semid does not exist)

`E_OACV` - Object access violation (A semid less than -4 was specified from a user task. This is implementation dependent.)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This routine deletes the semaphore specified by `semid`. All tasks blocked waiting to acquire the semaphore will be readied and returned a status code which indicates that the semaphore was deleted. The control block for this semaphore is reclaimed by RTEMS.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

The calling task will be preempted if it is enabled by the task's execution mode and a higher priority local task is waiting on the deleted semaphore. The calling task will NOT be preempted if all of the tasks that are waiting on the semaphore are remote tasks.

The calling task does not have to be the task that created the semaphore. Any local task that knows the semaphore id can delete the semaphore.

3.4.3 sig_sem - Signal Semaphore

CALLING SEQUENCE:

```
ER sig_sem(
  ID semid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (semid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the semaphore specified by semid does not exist)

`E_OACV` - Object access violation (A semid less than -4 was specified from a user task. This is implementation dependent.)

`E_QOVR` - Queuing or nesting overflow (the queuing count given by semcnt went over the maximum allowed)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

3.4.4 wai_sem - Wait on Semaphore

CALLING SEQUENCE:

```
ER wai_sem(
  ID semid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (semid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the semaphore specified by semid does not exist)

`E_OACV` - Object access violation (A semid less than -4 was specified from a user task. This is implementation dependent.)

`E_DLT` - The object being waited for was deleted (the specified semaphore was deleted while waiting)

`E_RLWAI` - Wait state was forcibly released (rel_wai was received while waiting)

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for tmout)

DESCRIPTION:

This routine attempts to acquire the semaphore specified by `semid`. If the semaphore is available (i.e. positive semaphore count), then the semaphore count is decremented and the calling task returns immediately. Otherwise the calling tasking is blocked until the semaphore is released by a subsequent invocation of `sig_sem`.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

If the semaphore is not available, then the calling task will be blocked.

3.4.5 preq_sem - Poll and Request Semaphore

CALLING SEQUENCE:

```
ER preq_sem(
  ID semid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (semid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the semaphore specified by semid does not exist)

`E_OACV` - Object access violation (A semid less than -4 was specified from a user task. This is implementation dependent.)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for tmout)

DESCRIPTION:

This routine attempts to acquire the semaphore specified by `semid`. If the semaphore is available (i.e. positive semaphore count), then the semaphore count is decremented and the calling task returns immediately. Otherwise, the `E_TMOUT` error is returned to the calling task to indicate the semaphore is unavailable.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

This routine will not cause the running task to be preempted.

3.4.6 twai_sem - Wait on Semaphore with Timeout

CALLING SEQUENCE:

```
ER twai_sem(
  ID semid,
  TMO tmout
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (semid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the semaphore specified by semid does not exist)

`E_OACV` - Object access violation (A semid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the specified semaphore was deleted while waiting)

`E_RLWAI` - Wait state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for tmout)

DESCRIPTION:

This routine attempts to acquire the semaphore specified by `semid`. If the semaphore is available (i.e. positive semaphore count), then the semaphore count is decremented and the calling task returns immediately. Otherwise the calling tasking is blocked until the semaphore is released by a subsequent invocation of `sig_sem` or the timeout period specified by `tmout` milliseconds is exceeded. If the timeout period is exceeded, then the `E_TMOUT` error is returned.

By specifiying `tmout` as `TMO_FEVR`, this routine has the same behavior as `wai_sem`. Similarly, by specifiying `tmout` as `TMO_POL`, this routine has the same behavior as `preq_sem`.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

This routine may cause the calling task to block.

A clock tick is required to support the timeout functionality of this routine.

3.4.7 ref_sem - Reference Semaphore Status

CALLING SEQUENCE:

```
ER ref_sem(
  T_RSEM *pk_rsem,
  ID semid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (semid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the semaphore specified by semid does not exist)

`E_OACV` - Object access violation (A semid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (the packet address for the return parameters could not be used)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_RPAR` - A value outside the range supported by the requesting node and/or transmission packet format was returned as a parameter (a value outside supported range was specified for exinf, wtsk or semcnt)

DESCRIPTION:

This routine returns status information on the semaphore specified by `semid`. The `pk_rsem` structure is filled in by this service call.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

This routine will not cause the running task to be preempted.

4 Eventflags Manager

4.1 Introduction

The eventflag manager provides a high performance method of intertask communication and synchronization. The directives provided by the eventflag manager are:

The services provided by the eventflags manager are:

- `cre_flg` - Create Eventflag
- `del_flg` - Delete Eventflag
- `set_flg` - Set Eventflag
- `clr_flg` - Clear Eventflag
- `wai_flg` - Wait on Eventflag
- `pol_flg` - Wait for Eventflag (Polling)
- `twai_flg` - Wait on Eventflag with Timeout
- `ref_flg` - Reference Eventflag Status

4.2 Background

4.2.1 Event sets

An eventflag is used by a task (or ISR) to inform another task of the occurrence of a significant situation. One word bit-field is associated with each eventflags. The application developer should remember the following key characteristics of event operations when utilizing the event manager:

- Events provide a simple synchronization facility.
- Events are aimed at tasks.
- Tasks can wait on more than one event simultaneously.
- Events are independent of one another.
- Events do not hold or transport data.
- Events are not queued. In other words, if an event is sent more than once to a task before being received, the second and subsequent send operations to that same task have no effect.

A pending event is an event that has been set. An event condition is used to specify the events which the task desires to receive and the algorithm which will be used to determine when the request is satisfied. An event condition is satisfied based upon one of two algorithms which are selected by the user. The `TWF_ORW` algorithm states that an event condition is satisfied when at least a single requested event is posted. The `TWF_ANDW` algorithm states that an event condition is satisfied when every requested event is posted.

An eventflags or condition is built by a bitwise OR of the desired events. If an event is not explicitly specified in the set or condition, then it is not present. Events are specifically designed to be mutually exclusive, therefore bitwise OR and addition operations are equivalent as long as each event appears exactly once in the event set list.

4.2.2 T_CFLG Structure

The T_CFLG structire is used to define the characteristics of an eventflag and passed as an argument to the `cre_flg` service. The structure is defined as follows:

```
/*
 * Create Eventflags (cre_flg) Structure
 */

typedef struct t_cflg {
  VP exinf;     /* extended information */
  ATR flgatr;   /* eventflag attribute */
  UINT iflgptn; /* initial eventflag */
  /* additional implementation dependent information may be included */
} T_CFLG;

/*
 *  flgatr - Eventflag Attribute Values
 */


/* multiple tasks are not allowed to wait (Wait Single Task)*/

#define TA_WSGL 0x00

/* multiple tasks are allowed to wait (Wait Multiple Task) */

#define TA_WMUL 0x08

/* wfmode */
#define TWF_ANDW 0x00 /* AND wait */
#define TWF_ORW 0x02 /* OR wait */
#define TWF_CLR 0x01 /* clear specification */
```

where the meaning of each field is:

exinf

may be used freely by the user for including extended information about the eventflag to be created. Information set here may be accessed by `ref_flg`. If a larger region is desired for including user information, or if the user wishes to change the contents of this information, the usr should allocate memory area and set the address of this memory packet to `exinf`. The OS does not take care of the contents of `exinf`. This implementation does not use this field.

flgatr

is the attributes for this eventflag. The lower bits of flgatr represent system attributes, while the upper bits represent implementation-dependent attributes.

iflgptn

is the initial eventflag pattern. (CPU and/or implementation-dependent information may also be included)

4.2.3 T_RFLG Structure

The T_RFLG structire is used to define the characteristics of an eventflag and passed as an argument to the `ref_flg` service. The structure is defined as follows:

```
/* Reference Eventflags (ref_flg) Structure */
typedef struct t_rflg {
 VP       exinf;  /* extended information */
 BOOL_ID  wtsk;   /* indicates whether or not there is a waiting task */
 UINT     flgptn; /* eventflag bit pattern */
 /* additional implementation dependent information may be included */
} T_RFLG;
```

exinf

see `T_CFLG`.

wtsk

indicates whether or not there is a task waiting for the eventflag in question. If there is no waiting task, `wtsk` is returned as FALSE = 0. If there is a waiting task, `wtsk` is returned as a value other than 0.

flgptn

is the eventflag pattern.

4.3 Operations

4.4 System Calls

This section details the eventflags manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

4.4.1 cre_flg - Create Eventflag

CALLING SEQUENCE:

```
ER cre_flg(
  ID flgid,
  T_CFLG *pk_cflg
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_NOMEM` - Insufficient memory (Memory for control block cannot be allocated)

`E_ID` - Invalid ID number (flgid was invalid or could not be used)

`E_RSATR` - Reserved attribute (flgatr was invalid or could not be used)

`E_OBJ` - Invalid object state (an eventflag of the same ID already exists)

`E_OACV` - Object access violation (A flgid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (pk_cflg is invalid)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_PAR`- A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for exinf, flgatr and/or iflgptn)

DESCRIPTION:

This system call creates the eventflag specified by `flgid`. Specifically, a control block for the eventflag to be created is allocated and the associated flag pattern is initialized using `iflgptn`. A single eventflag handles one word's worth of bits of the processor in question as a group. All operations are done in single word units.

User eventflags have positive ID numbers, while system eventflags have negative ID numbers. User tasks (tasks having positive task IDs) cannot access system eventflags. An `E_OACV` error will result if a user task issues a system call on a system eventflag, but error detection is implementation dependent.

Eventflags having ID numbers from -4 through 0 cannot be created. An `E_ID` error will result if a value in this range is specified for `flgid`.

The system attribute part of `flgatr` may be specified as `TA_WSGL` (Wait Single Task) or `TA_WMUL` (Wait Multiple Tasks)

NOTES:

Multiprocessing is not supported. Thus none of the `"EN_"` status codes will be returned.

All memory is preallocated for RTEMS ITRON objects. Thus, no dynamic memory allocation is performed by cre_flg and the E_NOMEM error can not be returned.

4.4.2 del_flg - Delete Eventflag

CALLING SEQUENCE:

```
ER del_flg(
  ID flgid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (flgid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the eventflag specified by flgid does not exist)

`E_OACV` - Object access violation (A flgid less than -4 was specified from a user task. This is implementation dependent.)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This system call deletes the eventflag specified by `flgid`.

Issuing this system call causes memory used for the control block of the associated eventflag to be released. After this system call is invoked, another eventflag having the same ID number can be created.

This system call will complete normally even if there are tasks waiting for the eventflag. In that case, an `E_DLT` error will be returned to each waiting task.

NOTES:

Multiprocessing is not supported. Thus none of the `"EN_"` status codes will be returned.

When an eventflag being waited for by more than one tasks is deleted, the order of tasks on the ready queue after the WAIT state is cleared is implementation dependent in the case of tasks having the same priority.

4.4.3 set_flg - Set Eventflag

CALLING SEQUENCE:

```
ER set_flg(
  ID flgid,
  UINT setptn
);
```

STATUS CODES:

E_OK - Normal Completion

E_ID - Invalid ID number (flgid was invalid or could not be used)

E_NOEXS - Object does not exist (the eventflag specified by flgid does not exist)

E_OACV - Object access violation (A flgid less than -4 was specified from a user task. This is implementation dependent.)

EN_OBJNO - An object number which could not be accessed on the target node is specified.

EN_CTXID - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

EN_PAR - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for setptn or clrptn)

DESCRIPTION:

The `set_flg` system call sets the bits specified by `setptn` of the one word eventflag specified by `flgid`. In other words, a logical sum is taken for the values of the eventflag specified by `flgid` with the value of `setptn`.

If the eventflag value is changed by `set_flg` and the new eventflag value satisfies the condition to release the WAIT state of the task which issued `wai_flg` on the eventflag, the WAIT state of that task will be released and the task will be put into RUN or READY state (or SUSPEND state if the task was in WAIT-SUSPEND).

Nothing will happen to the target eventflag if all bits of `setptn` are specified as 0 with `set_flg`. No error will result in either case.

Multiple tasks can wait for a single eventflag if that eventflags has the `TA_WMUL` attribute. This means that even eventflags can make queues for tasks to wait on. When such eventflags are used, a single `set_flg` call may result in the release of multiple waiting tasks. In this case, the order of tasks on the ready queue after the WAIT state is cleared is preserved for tasks having the same priority.

NOTES:

Multiprocessing is not supported. Thus none of the `"EN_"` status codes will be returned.

4.4.4 clr_flg - Clear Eventflag

CALLING SEQUENCE:

```
ER clr_flg(
  ID flgid,
  UINT clrptn
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (flgid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the eventflag specified by flgid does not exist)

`E_OACV` - Object access violation (A flgid less than -4 was specified from a user task. This is implementation dependent.)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for setptn or clrptn)

DESCRIPTION:

The `clr_flg` system call clears the bits of the one word eventflag based on the corresponding zero bits of `clrptn`. In other words, a logical product is taken for the values of the eventflag specified by `flgid` with the value of `clrptn`.

Issuing `clr_flg` never results in wait conditions being released on a task waiting for the specified eventflag. In other words, dispatching never occurs with `clr_flg`.

Nothing will happen to the target eventflag if all bits of `clrptn` are specified as 1 with `clr_flg`. No error will result.

NOTES:

Multiprocessing is not supported. Thus none of the `"EN_"` status codes will be returned.

4.4.5 wai_flg - Wait on Eventflag

CALLING SEQUENCE:

```
ER wai_flg(
  UINT *p_flgptn,
  ID flgid,
  UINT waiptn,
  UINT wfmode
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (flgid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the eventflag specified by flgid does not exist)

`E_OACV` - Object access violation (A flgid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (waiptn = 0, wfmode invalid, or tmout is -2 or less)

`E_OBJ` - Invalid object state (multiple tasks waiting for an eventflag with the TA_WSGL attribute)

`E_DLT` - The object being waited for was deleted (the specified eventflag was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for waiptn and tmout)

`EN_RPAR` - A value outside the range supported by the requesting node and/or transmission packet format was specified as a parameter (a value exceeding the range for the requesting node was specified for flgptn)

DESCRIPTION:

The `wai_flg` system call waits for the eventflag specified by `flgid` to be set to satisfy the wait release condition specified by `wfmode`. The Eventflags bit-pattern will be returned with a pointer `p_flgptn`.

If the eventflag specified by `flgid` already satisfies the wait release conditions given by `wfmode`, the issuing task will continue execution without waiting. `wfmode` may be specified as follows.

```
wfmode = TWF_ANDW (or TWF_ORW) | TWF_CLR(optional)
```

If TWF_ORW is specified, the issuing task will wait for any of the bits specified by waiptn to be set for the eventflag given by flgid (OR wait). If TWF_ANDW is specified, the issuing task will wait for all of the bits specified by waiptn to be set for the eventflag given by flgid (AND wait).

If the TWF_CLR specification is not present, the eventflag value will remain unchanged even after the wait conditions have been satisfied and the task has been released from the WAIT state. If TWF_CLR is specified, all bits of the eventflag will be cleared to 0 once the wait conditions of the waiting task have been satisfied.

The return parameter flgptn returns the value of the eventflag after the wait state of a task has been released due to this system call. If TWF_CLR was specified, the value before eventflag bits were cleared is returned. The value returned by flgptn fulfills the wait release conditions of this system call.

An E_PAR parameter error will result if waiptn is 0.

A task can not execute any of wai_flg, twai_flg or pol_flg on an eventflag having the TA_WSGL attribute if another task is already waiting for that eventflag. An E_OBJ error will be returned to a task which executes wai_flg at a later time regardless as to whether or not the task that executes wai_flg or twai_flg later will be placed in a WAIT state (conditions for releasing wait state be satisfied). An E_OBJ error will be returned even to tasks which just execute pol_flg, again regardless as to whether or not wait release conditions for that task were satisfied.

On the other hand, multiple tasks can wait at the same time for the same eventflag if that eventflag has the TA_WMUL attribute. This means that event flags can make queues for tasks to wait on. When such eventflags are used, a single set_flg call may release multiple waiting tasks.

The following processing takes place if a queue for allowing multiple tasks to wait has been created for an eventflag with the TA_WMUL attribute.

- The waiting order on the queue is FIFO. (However, depending on waiptn and wfmode, task at the head of the queue will not always be released from waiting.)
- If a task specifying that the eventflag be cleared is on the queue, the flag is cleared when that task is released from waiting.
- Since any tasks behind a task which clears the eventflag (by specifying TWF_CLR) will check the eventflag after it is cleared, they will not be released from waiting.

If multiple tasks having the same priority are released from waiting simultaneously due to set_flg, the order of tasks on the ready queue after release will be the same as their original order on the eventflag queue.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

4.4.6 pol_flg - Wait for Eventflag (Polling)

CALLING SEQUENCE:

```
ER pol_flg(
  UINT *p_flgptn,
  ID flgid,
  UINT waiptn,
  UINT wfmode
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (flgid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the eventflag specified by flgid does not exist)

`E_OACV` - Object access violation (A flgid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (waiptn = 0, wfmode invalid, or tmout is -2 or less)

`E_OBJ` - Invalid object state (multiple tasks waiting for an eventflag with the TA_WSGL attribute)

`E_DLT` - The object being waited for was deleted (the specified eventflag was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for waiptn and tmout)

`EN_RPAR` - A value outside the range supported by the requesting node and/or transmission packet format was specified as a parameter (a value exceeding the range for the requesting node was specified for flgptn)

DESCRIPTION:

The `pol_flg` system call has the same function as `wai_flg` except for the waiting feature. `pol_flg` polls whether or not the task should wait if `wai_flg` is executed. The meanings of parameters to `pol_flg` are the same as for `wai_flg`. The specific operations by `pol_flg` are as follows.

- If the target eventflag already satisfies the conditions for releasing wait given by `wfmode`, processing is the same as `wai_flg`: the eventflag is cleared if `TWF_CLR` is specified and the system call completes normally.

- If the target eventflag does not yet satisfy the conditions for releasing wait given by `wfmode`, an `E_TMOUT` error is returned to indicate polling failed and the system call finishes. Unlike `wai_flg`, the issuing task does not wait in this case. The eventflag is not cleared in this case even if `TWF_CLR` has been specified.

NOTES:

Multiprocessing is not supported. Thus none of the `"EN_"` status codes will be returned.

4.4.7 twai_flg - Wait on Eventflag with Timeout

CALLING SEQUENCE:

```
ER twai_flg(
  UINT *p_flgptn,
  ID flgid,
  UINT waiptn,
  UINT wfmode,
  TMO tmout
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (flgid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the eventflag specified by flgid does not exist)

`E_OACV` - Object access violation (A flgid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (waiptn = 0, wfmode invalid, or tmout is -2 or less)

`E_OBJ` - Invalid object state (multiple tasks waiting for an eventflag with the TA_WSGL attribute)

`E_DLT` - The object being waited for was deleted (the specified eventflag was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for waiptn and tmout)

`EN_RPAR` - A value outside the range supported by the requesting node and/or transmission packet format was specified as a parameter (a value exceeding the range for the requesting node was specified for flgptn)

DESCRIPTION:

The `twai_flg` system call has the same function as `wai_flg` with an additional timeout feature. A maximum wait time (timeout value) can be specified using the parameter `tmout`. When a timeout is specified, a timeout error, `E_TMOUT`, will result and the system call will finish if the period specified by `tmout` elapses without conditions for releasing wait being satisfied.

Specifying `TMO_POL = 0` to `twai_flg` for `tmout` indicates that a timeout value of 0 be used, resulting in exactly the same processing as `pol_flg`. Specifying `TMO_FEVR = -1` to `twai_flg` for `tmout` indicates that an infinite timeout value be used, resulting in exactly the same processing as `wai_flg`.

NOTES:

Multiprocessing is not supported. Thus none of the `"EN_"` status codes will be returned.

`Pol_flg` and `wai_flg` represent the same processing as specifying certain values (`TMO_POL` or `TMO_FEVR`) to `twai_flg` for tmout. As such, only `twai_flg` is implemented in the kernel; `pol_flg` and `wai_flg` should be implemented as macros which call `twai_flg`.

4.4.8 ref_flg - Reference Eventflag Status

CALLING SEQUENCE:

```
ER ref_flg(
  T_RFLG *pk_rflg,
  ID flgid );
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (flgid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the eventflag specified by flgid does not exist)

`E_OACV` - Object access violation (A flgid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (the packet address for the return parameters could not be used)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_RPAR` - A value outside the range supported by the requesting node and/or transmission packet format was returned as a parameter (a value outside supported range was specified for exinf, wtsk and/or flgptn)

DESCRIPTION:

This system call refers to the state of the eventflag specified by `flgid`, and returns its current flag pattern (`flgptn`), waiting task information (`wtsk`), and its extended information (`exinf`).

Depending on the implementation, `wtsk` may be returned as the ID (non-zero) of the task waiting for the eventflag. If there are multiple tasks waiting for the eventflag (only when attribute is `TA_WMUL`), the ID of the task at the head of the queue is returned.

An `E_NOEXS` error will result if the eventflag specified to `ref_flg` does not exist.

NOTES:

Multiprocessing is not supported. Thus none of the `"EN_"` status codes will be returned.

Although both `ref_flg` and `pol_flg` can be used to find an eventflag's pattern (`flgptn`) without causing the issuing task to wait, `ref_flg` simply reads the eventflag's pattern (`flgptn`) while `pol_flg` functions is identical to `wai_flg` when wait release conditions are satisfied (it clears the eventflag if `TWF_CLR` is specified).

Depending on the implementation, additional information besides `wtsk` and `flgptn` (such as eventflag attributes, `flgatr`) may also be referred.

5 Mailbox Manager

5.1 Introduction

The mailbox manager is basically a linked list, hidden by the super core message queue and consists of a control block, a private structure. The control block comprises of the create mailbox structure, the message structure and the reference mailbox structure.

The services provided by the mailbox manager are:

- `cre_mbx` - Create Mailbox
- `del_mbx` - Delete Mailbox
- `snd_msg` - Send Message to Mailbox
- `rcv_msg` - Receive Message from Mailbox
- `prcv_msg` - Poll and Receive Message from Mailbox
- `trcv_msg` - Receive Message from Mailbox with Timeout
- `ref_mbx` - Reference Mailbox Status

5.2 Background

5.3 Operations

5.4 System Calls

This section details the mailbox manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

5.4.1 cre_mbx - Create Mailbox

CALLING SEQUENCE:

```
ER cre_mbx(
  ID      mbxid,
  T_CMBX *pk_cmbx
);
```

STATUS CODES:

`E_OK` - Normal completion
`E_NOMEM` - Insufficient memory
`E_ID` - Invalid ID number
`E_RSATR` - Reserved attribute
`E_OBJ` - Invalid object state
`E_OACV` - Object access violation
`E_PAR` - Parameter error

DESCRIPTION:

Allocated a control area/buffer space for mailbox with some ID.

```
User area:   +ve ids
System area:   -ve ids
```

User may specify if its FIFO or priority level queue. Assumes shared memory b/w communicating processes. Initializes core message queue for this mbox.

NOTES:

NONE

5.4.2 del_mbx - Delete Mailbox

CALLING SEQUENCE:

```
ER del_mbx(
  ID mbxid
);
```

STATUS CODES:

`E_OK` - Normal completion
`E_ID` - Invalid ID number
`E_NOEXS` - Object does not exist
`E_OACV` - Object access violation

DESCRIPTION:

Specified by the ID, cleans up all data structures and control blocks.

NOTES:

NONE

5.4.3 snd_msg - Send Message to Mailbox

CALLING SEQUENCE:

```
ER snd_msg(
  ID     mbxid,
  T_MSG *pk_msg
);
```

STATUS CODES:

`E_OK` - Normal completion
`E_ID` - Invalid ID number
`E_NOEXS` - Object does not exist
`E_OACV` - Object access violation
`E_QOVR` - Queueing or nesting overflow

DESCRIPTION:

Sends the address of message to mbox having a given id, any waiting tasks (blocked tasks) will be woken up. It supports non-blocking send.

NOTES:

NONE

5.4.4 rcv_msg - Receive Message from Mailbox

CALLING SEQUENCE:

```
ER rcv_msg(
  T_MSG **ppk_msg,
  ID      mbxid
);
```

STATUS CODES:

E_OK - Normal completion
E_ID - Invalid ID number
E_NOEXS - Object does not exist
E_OACV - Object access violation
E_PAR - Parameter error
E_DLT - The object being waited for was deleted
E_RLWAI - WAIT state was forcibly released
E_CTX - Context error

DESCRIPTION:

If there is no message then receiver blocks, if not empty then it takes the first message of the queue.

NOTES:

NONE

5.4.5 prcv_msg - Poll and Receive Message from Mailbox

CALLING SEQUENCE:

```
ER prcv_msg(
  T_MSG **ppk_msg,
  ID      mbxid
);
```

STATUS CODES:

`E_OK` - Normal completion
`E_ID` - Invalid ID number
`E_NOEXS` - Object does not exist
`E_OACV` - Object access violation
`E_PAR` - Parameter error
`E_DLT` - The object being waited for was deleted
`E_RLWAI` - WAIT state was forcibly released
`E_CTX` - Context error

DESCRIPTION:

Poll and receive message from mailbox.

NOTES:

NONE

5.4.6 trcv_msg - Receive Message from Mailbox with Timeout

CALLING SEQUENCE:

```
ER trcv_msg(
  T_MSG **ppk_msg,
  ID      mbxid,
  TMO     tmout
);
```

STATUS CODES:

`E_OK` - Normal completion
`E_ID` - Invalid ID number
`E_NOEXS` - Object does not exist
`E_OACV` - Object access violation
`E_PAR` - Parameter error
`E_DLT` - The object being waited for was deleted
`E_RLWAI` - WAIT state was forcibly released
`E_CTX` - Context error

DESCRIPTION:

Blocking receive with a maximum timeout.

NOTES:

NONE

5.4.7 ref_mbx - Reference Mailbox Status

CALLING SEQUENCE:

```
ER ref_mbx(
  T_RMBX *pk_rmbx,
  ID      mbxid
);
```

STATUS CODES:

`E_OK` - Normal completion
`E_ID` - Invalid ID number
`E_NOEXS` - Object does not exist
`E_OACV` - Object access violation
`E_PAR` - Parameter error

DESCRIPTION:

Supports non-blocking receive. If there are no messages, it returns -1. Also returns id of the next process waiting on a message.

NOTES:

NONE

6 Message Buffer Manager

6.1 Introduction

The message buffer manager provides functions to create, delete, and control of message buffers. This manager is based on the ITRON 3.0 standard.

The services provided by the message buffer manager are:

- `cre_mbf` - Create MessageBuffer
- `del_mbf` - Delete MessageBuffer
- `snd_mbf` - Send Message to MessageBuffer
- `psnd_mbf` - Poll and Send Message to MessageBuffer
- `tsnd_mbf` - Send Message to MessageBuffer with Timeout
- `rcv_mbf` - Receive Message from MessageBuffer
- `prcv_mbf` - Poll and Receive Message from MessageBuffer
- `trcv_mbf` - Receive Message from MessageBuffer with Timeout
- `ref_mbf` - Reference MessageBuffer Status

6.2 Background

6.2.1 T_CMBF Structure

The T_CMBF structure is used to define the characteristics of a message buffer and passed as an argument to the `cre_mbf` routine. This structure is defined as:

```
typedef struct t_cmbf {
  VP      exinf;   /* extended information */
  ATR     mbfatr;  /* message buffer attributes */
  INT     bufsz;   /* buffer size (in bytes) */
  INT     maxmsz;  /* maximum message size (in bytes) */
  /* (CPU and/or implementation-dependent information may also be
  included) */
} T_CMBF;
```

where the meaning of each field is:

exinf is for any extended information that the implementation may define. This implementation does not use this field.

mbfatr is the attributes for the message buffer. The only attributes which can be specified is whether tasks wait in FIFO (`TA_TFIFO`) or priority (`TA_TPRI`) order.

bufsz is the size of the message buffer. Since some control data are needed to manage each messages in the message buffer, `bufsz` is usually larger than the total of all message sizes in this buffer.

maxmsz is the maximum message size.

6.2.2 T_RMBF Structure

The T_RMBF structure is filled in by the `ref_mbf` routine with status and state information of the message buffer. The structure is defined as follows:

```
typedef struct t_rmbf {
  VP      exinf;   /* extended information */
  BOOL_ID wtsk;    /* waiting task information */
  BOOL_ID stsk;    /* sending task information */
  INT     msgsz;   /* message size (in bytes) */
  INT     frbufsz; /* free buffer size (in bytes) */
  /* (CPU and/or implementation-dependent information is returned) */
} T_RMBF;
```

exinf is for any extended information that the implementation may define. This implementation does not use this field.

wtsk is TRUE when there is one or more tasks waiting on this message buffer to send message. It is FALSE when there is no waiting task. The meaning of this field is allowed to vary between ITRON implementations. It may have the ID of a waiting task, the number of tasks waiting, or a boolean indication that one or more tasks are waiting.

stsk is TRUE when there is one or more tasks waiting on this message buffer to receive message. It is FALSE when there is no waiting task. The meaning of this field is allowed to vary between ITRON implementations. It may have the ID of a waiting task, the number of tasks waiting, or a boolean indication that one or more tasks are waiting.

msgsz is the size of the message that is at the head of the message buffer. If there is no message on the message queue, `msgsz` will be returned as FALSE = 0.

frbufsz is the amount of free memory in the message buffer.

6.3 Operations

6.4 System Calls

This section details the message buffer manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

6.4.1 cre_mbf - Create MessageBuffer

CALLING SEQUENCE:

```
ER cre_mbf(
  ID mbfid,
  T_CMBF *pk_cmbf
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_NOMEM` - Insufficient memory (Memory for control block and/or ring buffer cannot be allocted)

`E_ID` - Invalid ID number (mbfid was invalid or could not be used)

`E_RSATR` - Reserved attribute (mbfatr was invalid or could not be used)

`E_OBJ` - Invalid object state (a messagebuffer of the same ID already exists)

`E_OACV` - Object access violation (A mbfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (pk_cmbf is invalid or bufsz and/or maxmsz is negative or invalid)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for exinf, mbfatr, bufsz and/or maxmsz)

DESCRIPTION:

This routine creates a message buffer on the local node. The message buffer is initialized based on the attributes specified in the `pk_cmbf` structure. The buffer size and the maximum message size are determined by the `bufsz` and `maxmsz` fields in this structure.

The `mbfatr` field represents attributes of the message buffer. If `TA_TFIFO` is specified, tasks will be put on the queue on a First In-First Out basis. If `TA_TPRI` is specified, tasks will be placed on the queue according to their priority.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

6.4.2 del_mbf - Delete MessageBuffer

CALLING SEQUENCE:

```
ER del_mbf(
  ID mbfid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mbfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the messagebuffer specified by mbfid does not exist)

`E_OACV` - Object access violation (A mbfid less than -4 was specified from a user task. This is implementation dependent.)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

DESCRIPTION:

This routine deletes the message buffer specified by `mbfid`. Issuing this system call releases memory area used for the control block of the associated message buffer and the buffer area used for storing messages.

This routine will complete normally even if there are tasks waiting to send or receive messages at the message buffer. In that case, an `E_DLT` error will be returned to each waiting task. If there are messages still in the message buffer, they will be deleted along with the message buffer and no error will result.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

6.4.3 snd_mbf - Send Message to Message Buffer

CALLING SEQUENCE:

```
ER snd_mbf(
  ID mbfid,
  VP msg,
  INT msgsz
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mbfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the message buffer specified by mbfid does not exist)

`E_OACV` - Object access violation (A mbfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (msgsz is 0 or less; msgsz is larger than maxmsz; values unsuitable for msg; tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the message buffer of interest was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state; implementation dependent for psnd_mbf and tsnd_mbf(tmout=TMO_POL))

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion (implementation-dependent; applicable to psnd_mbf and tsnd_mbf (tmout=TMO_POL) only)

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for msgsz and/or tmout)

DESCRIPTION:

This routine sends the message stored at the address given by `msg` to the message buffer specified by `mbfid`. The size of the message is specified by `msgsz`; that is, `msgsz` number of bytes beginning from `msg` are copied to the message buffer specified by `mbfid`. If the available space in the buffer is not enough to include the message given by `msg`, the task issuing this system call will wait on a send wait queue until more buffer space becomes available.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

6.4.4 psnd_mbf - Poll and Send Message to Message Buffer

CALLING SEQUENCE:

```
ER ercd =psnd_mbf(
  ID mbfid,
  VP msg,
  INT msgsz
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mbfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the message buffer specified by mbfid does not exist)

`E_OACV` - Object access violation (A mbfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (msgsz is 0 or less; msgsz is larger than maxmsz; values unsuitable for msg; tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the message buffer of interest was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state; implementation dependent for psnd_mbf and tsnd_mbf(tmout=TMO_POL))

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion (implementation-dependent; applicable to psnd_mbf and tsnd_mbf (tmout=TMO_POL) only)

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for msgsz and/or tmout)

DESCRIPTION:

This routine has the same function as `snd_mbf` except for the waiting feature. `Psnd_mbf` polls whether or not the task should wait if `snd_mbf` is executed. The meaning of parameters to `psnd_mbf` are the same as for `snd_mbf`. The specific operations by `psnd_mbf` are as follows.

- If there is enough space in the buffer, processing is the same as `snd_mbf`: the message is sent and the system call completes normally.

- If there is not enough space in the buffer, an `E_TMOUT` error is returned to indicate polling failed and the system call finishes. Unlike `snd_mbf`, the issuing task does not wait in this case. The status of the message buffer and the message queue remain unchanged.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

6.4.5 tsnd_mbf - Send Message to Message Buffer with Timeout

CALLING SEQUENCE:

```
ER ercd =tsnd_mbf(
  ID mbfid,
  VP msg,
  INT msgsz,
  TMO tmout
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mbfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the message buffer specified by mbfid does not exist)

`E_OACV` - Object access violation (A mbfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (msgsz is 0 or less; msgsz is larger than maxmsz; values unsuitable for msg; tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the message buffer of interest was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state; implementation dependent for psnd_mbf and tsnd_mbf(tmout=TMO_POL))

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion (implementation-dependent; applicable to psnd_mbf and tsnd_mbf (tmout=TMO_POL) only)

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for msgsz and/or tmout)

DESCRIPTION:

The `tsnd_mbf` system call has the same function as `snd_mbf` with an additional timeout feature. A maximum wait time (timeout value) can be specified using the parameter `tmout`. When a timeout is specified, a timeout error, `E_TMOUT`, will result and the system call will finish if the period specified by `tmout` elapses without conditions for releasing wait being satisfied (i.e. without sufficient buffer space becoming available).

Only positive values can be specified for `tmout`. Specifying `TMO_POL` = 0 to `tsnd_mbf` for `tmout` indicates that a timeout value of 0 be used, resulting in exactly the same processing as psnd_mbf. Specifying TMO_FEVR = -1 to tsnd_mbf for tmout indicates that an infinite timeout value be used, resulting in exactly the same processing as snd_mbf.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

6.4.6 rcv_mbf - Receive Message from Message Buffer

CALLING SEQUENCE:

```
ER rcv_mbf(
  VP msg,
  INT *p_msgsz,
  ID mbfid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mbfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the message buffer specified by mbfid does not exist)

`E_OACV` - Object access violation (A mbfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (values unsuitable for msg; tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the specified message buffer was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for tmout)

`EN_RPAR` - A value outside the range supported by the issuing node and/or transmission packet format was returned as a return parameter (msgsz is outside supported range for requesting node)

DESCRIPTION:

`Rcv_mbf` receives the message from the message buffer specified by `mbfid`, and stores it at the memory location given by `msg`. In other words, the content of the message at the head of the message buffer specified by `mbfid` is copied into an area which begins from `msg` and whose size is `msgsz`.

If the message buffer is empty, the task issuing this system call will wait on a receive message wait queue until a message arrives.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

6.4.7 prcv_mbf - Poll and Receive Message from Message Buffer

CALLING SEQUENCE:

```
ER ercd =prcv_mbf(
  VP msg,
  INT *p_msgsz,
  ID mbfid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mbfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the message buffer specified by mbfid does not exist)

`E_OACV` - Object access violation (A mbfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (values unsuitable for msg; tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the specified message buffer was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for tmout)

`EN_RPAR` - A value outside the range supported by the issuing node and/or transmission packet format was returned as a return parameter (msgsz is outside supported range for requesting node)

DESCRIPTION:

The `prcv_mbf` system call has the same function as `rcv_mbf` except for the waiting feature. `Prcv_mbf` polls whether or not the task should wait if `rcv_mbf` is executed. The meaning of parameters to `prcv_mbf` are the same as for `rcv_mbf`. The specific operations by `prcv_mbf` are as follows.

- If there is a message in the specified message buffer, processing is the same as `rcv_mbf`: the first message on the message buffer is retrieved and the system call completes normally.

- If there is no message in the specified message buffer, an `E_TMOUT` error is returned to indicate polling failed and the system call finishes. Unlike `rcv_mbf`, the issuing task does not wait in this case. The status of the message buffer remain unchanged.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

6.4.8 trcv_mbf - Receive Message from Message Buffer with Timeout

CALLING SEQUENCE:

```
ER ercd =trcv_mbf(
  VP msg,
  INT *p_msgsz,
  ID mbfid,
  TMO tmout
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mbfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the message buffer specified by mbfid does not exist)

`E_OACV` - Object access violation (A mbfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (values unsuitable for msg; tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the specified message buffer was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_PAR` - A value outside the range supported by the target node and/or transmission packet format was specified as a parameter (a value outside supported range was specified for tmout)

`EN_RPAR` - A value outside the range supported by the issuing node and/or transmission packet format was returned as a return parameter (msgsz is outside supported range for requesting node)

DESCRIPTION:

The `trcv_mbf` system call has the same function as `rcv_mbf` with an additional timeout feature. A maximum wait time (timeout value) can be specified using the parameter `tmout`. When a timeout is specified, a timeout error, `E_TMOUT`, will result and the system call will finish if the period specified by `tmout` elapses without conditions for releasing wait being satisfied (i.e. without a message arriving).

Only positive values can be specified for `tmout`. Specifying `TMO_POL` = 0 to `trcv_mbf` for `tmout` indicates that a timeout value of 0 be used, resulting in exactly the same processing as `prcv_mbf`. Specifying `TMO_FEVR` = -1 to `trcv_mbf` for tmout indicates that an infinite timeout value be used, resulting in exactly the same processing as `rcv_mbf`.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

6.4.9 ref_mbf - Reference Message Buffer Status

CALLING SEQUENCE:

```
ER ref_mbf(
  T_RMBF *pk_rmbf,
  ID mbfid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mbfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the message buffer specified by mbfid does not exist)

`E_OACV` - Object access violation (A mbfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (the packet address for the return parameters could not be used)

`EN_OBJNO` - An object number which could not be accessed on the target node is specified.

`EN_CTXID` - Specified an object on another node when the system call was issued from a task in dispatch disabled state or from a task-independent portion

`EN_RPAR` - A value outside the range supported by the issuing node and/or transmission packet format was returned as a return parameter (a value outside supported range for exinf, wtsk, stsk, msgsz and/or frbufsz on a requesting node)

DESCRIPTION:

This system call refers to the state of the message buffer specified by `mbfid`, and returns information of a task waiting to send a message (`stsk`), the size of the next message to be received (`msgsz`), the free buffer size (`frbufsz`), information of a task waiting to receive a message (`wtsk`), and its extended information (`exinf`).

`Wtsk` and `stsk` indicate whether or not there is a task waiting for the message buffer in question. If there is no waiting task, `wtsk` and `stsk` are returned as `FALSE` = 0. If there is a waiting task, `wtsk` and `stsk` are returned as values other than 0.

An `E_NOEXS` error will result if the message buffer specified to `ref_mbf` does not exist.

The size of the message at the head of the message buffer (the next message to be received) is returned to `msgsz`. If there are no messages on the message buffer, `msgsz` will be returned as `FALSE` = 0. A message whose size is zero cannot be sent.

At least one of `msgsz` = `FALSE` and `wtsk` = `FALSE` is always true in this system call.

`Frbufsz` indicates the amount of free memory in the message buffer. This value can be used to know the total approximate size of the messages which can be sent to the message buffer.

NOTES:

Multiprocessing is not supported. Thus none of the "EN_" status codes will be returned.

7 Rendezvous Manager

7.1 Introduction

The rendezvous manager is ...

The services provided by the rendezvous manager are:

- `cre_por` - Create Port for Rendezvous
- `del_por` - Delete Port for Rendezvous
- `cal_por` - Call Port for Rendezvous
- `pcal_por` - Poll and Call Port for Rendezvous
- `tcal_por` - Call Port for Rendezvous with Timeout
- `acp_por` - Accept Port for Rendezvous
- `pacp_por` - Poll and Accept Port for Rendezvous
- `tacp_por` - Accept Port for Rendezvous with Timeout
- `fwd_por` - Forward Rendezvous to Other Port
- `rpl_rdv` - Reply Rendezvous
- `ref_por` - Reference Port Status

7.2 Background

7.3 Operations

7.4 System Calls

This section details the rendezvous manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

7.4.1 cre_por - Create Port for Rendezvous

CALLING SEQUENCE:

```
ER cre_por(
  ID porid,
  T_CPOR *pk_cpor
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

7.4.2 del_por - Delete Port for Rendezvous

CALLING SEQUENCE:

```
ER del_por(
  ID porid
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

7.4.3 cal_por - Call Port for Rendezvous Poll

CALLING SEQUENCE:

```
ER cal_por(
  VP msg,
  INT *p_rmsgsz,
  ID porid,
  UINT calptn
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

7.4.4 pcal_por - Poll and Call Port for Rendezvous

CALLING SEQUENCE:

```
ER ercd =pcal_por(
  VP msg,
  INT *p_rmsgsz,
  ID porid,
  UINT calptn,
  INT
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

7.4.5 tcal_por - Call Port for Rendezvous with Timeout

CALLING SEQUENCE:

```
ER ercd =tcal_por(
  VP msg,
  INT *p_rmsgsz,
  ID porid,
  UINT calptn,
  INT
);
```

STATUS CODES:

`EXXX` -

DESCRIPTION:

NOTES:

7.4.6 acp_por - Accept Port for Rendezvous Poll

CALLING SEQUENCE:

```
ER acp_por(
  RNO *p_rdvno,
  VP msg,
  INT *p_cmsgsz,
  ID porid
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

7.4.7 pacp_por - Poll and Accept Port for Rendezvous

CALLING SEQUENCE:

```
ER ercd =pacp_por(
  RNO *p_rdvno,
  VP msg,
  INT *p_cmsgsz,
  ID porid,
  UINT
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

7.4.8 tacp_por - Accept Port for Rendezvous with Timeout

CALLING SEQUENCE:

```
ER ercd =tacp_por(
  RNO *p_rdvno,
  VP msg,
  INT *p_cmsgsz,
  ID porid,
  UINT
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

7.4.9 fwd_por - Forward Rendezvous to Other Port

CALLING SEQUENCE:

```
ER fwd_por(
  ID porid,
  UINT calptn,
  RNO rdvno,
  VP msg,
  INT cmsgsz
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

7.4.10 rpl_rdv - Reply Rendezvous

CALLING SEQUENCE:

```
ER rpl_rdv(
  RNO rdvno,
  VP msg,
  INT rmsgsz
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

7.4.11 ref_por - Reference Port Status

CALLING SEQUENCE:

```
ER ref_por(
  T_RPOR *pk_rpor,
  ID porid
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

8 Interrupt Manager

8.1 Introduction

The interrupt manager is ...

The services provided by the interrupt manager are:

- `def_int` - Define Interrupt Handler
- `ret_int` - Return from Interrupt Handler
- `ret_wup` - Return and Wakeup Task
- `loc_cpu` - Lock CPU
- `unl_cpu` - Unlock CPU
- `dis_int` - Disable Interrupt
- `ena_int` - Enable Interrupt
- `chg_iXX` - Change Interrupt Mask(Level or Priority)
- `ref_iXX` - Reference Interrupt Mask(Level or Priority)

8.2 Background

8.3 Operations

8.4 System Calls

This section details the interrupt manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

8.4.1 def_int - Define Interrupt Handler

CALLING SEQUENCE:

```
ER def_int(
  UINT dintno,
  T_DINT *pk_dint
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

8.4.2 ret_int - Return from Interrupt Handler

CALLING SEQUENCE:

```
void ret_int(

);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

8.4.3 ret_wup - Return and Wakeup Task

CALLING SEQUENCE:

```
void ret_wup(
  ID tskid
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

8.4.4 loc_cpu - Lock CPU

CALLING SEQUENCE:

```
ER loc_cpu(

);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

8.4.5 unl_cpu - Unlock CPU

CALLING SEQUENCE:

```
ER unl_cpu(

);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

8.4.6 dis_int - Disable Interrupt

CALLING SEQUENCE:

```
ER dis_int(
  UINT eintno
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

8.4.7 ena_int - Enable Interrupt

CALLING SEQUENCE:

```
ER ena_int(
  UINT eintno
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

8.4.8 chg_iXX - Change Interrupt Mask(Level or Priority)

CALLING SEQUENCE:

```
ER chg_iXX(
  UINT iXXXX
);
```

STATUS CODES:

`EXXX` -

DESCRIPTION:

NOTES:

8.4.9 ref_iXX - Reference Interrupt Mask(Level or Priority)

CALLING SEQUENCE:

```
ER ref_iXX(
  UINT *p_iXXXX
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

9 Memory Pool Manager

9.1 Introduction

The memory pool manager is ...

The services provided by the memory pool manager are:

- `cre_mpl` - Create Variable-Size Memorypool
- `del_mpl` - Delete Variable-Size Memorypool
- `get_blk` - Get Variable-Size Memory Block
- `pget_blk` - Poll and Get Variable-Size Memory Block
- `tget_blk` - Get Variable-Size Memory Block with Timeout
- `rel_blk` - Release Variable-Size Memory Block
- `ref_mpl` - Reference Variable-Size Memorypool Status

9.2 Background

Memorypool management functions manage memorypools and allocate memory blocks. There are two types of memorypool: fixed-size memorypools and variable-size memorypools. Both are considered separate objects and require different system calls for manipulation. While the size of memory blocks allocated from fixed-size memorypools are all fixed, blocks of any size may be specified when allocating memory blocks from variable-size memorypools.

9.3 Operations

9.4 System Calls

This section details the memory pool manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

9.4.1 cre_mpl - Create Variable-Size Memorypool

CALLING SEQUENCE:

```
ER cre_mpl(
  ID mplid,
  T_CMPL *pk_cmpl
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_NOMEM` - Insufficient memory (Memory for control block and/or for memorypool cannot be allocated)

`E_ID` - Invalid ID number (mplid was invalid or could not be used)

`E_RSATR` - Reserved attribute (mplatr was invalid or could not be used)

`E_OBJ` - Invalid object state (a memorypool of the same ID already exists)

`E_OACV` - Object access violation (A mplid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (pk_cmpl is invalid or mplsz is negative or invalid)

DESCRIPTION:

The cre_mpl directive creates a variable-size memorypool having the ID number specified by mplid. Specifically, a memory area of the size determined by mplsz is allocated for use as a memorypool. A control block for the memorypool being created is also allocated. User memorypools have positive ID numbers, while system memorypools have negative ID numbers. User tasks (tasks having positive task IDs) cannot access system memorypools (memorypools having negative ID numbers).

NOTES:

The memory required for creating memorypools and for allocating control blocks for each object is reserved while system initialization. Associated parameters are therefore specified at system configuration.

9.4.2 del_mpl - Delete Variable-Size Memorypool

CALLING SEQUENCE:

```
ER del_mpl(
  ID mplid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mplid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mplid does not exist)

`E_OACV` - Object access violation (A mplid less than -4 was specified from a user task. This is implementation dependent.)

DESCRIPTION:

This system call deletes the variable-size memorypool specified by mplid. No check or error report is performed even if there are tasks using memory from the memorypool to be deleted. This system call completes normally even if some of the memory blocks are not returned. Issuing this system call causes memory used for the control block of the associated memorypool and the memory area making up the memorypool itself to be released. After this system call is invoked, another memorypool having the same ID number can be created.

NOTES:

When a memorypool being waited for by more than one tasks is deleted, the order of tasks on the ready queue after the WAIT state is cleared is implementation dependent in the case of tasks having the same priority.

9.4.3 get_blk - Get Variable-Size Memory Block

CALLING SEQUENCE:

```
ER get_blk(
  VP *p_blk,
  ID mplid,
  INT blksz
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mplid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mplid does not exist)

`E_OACV` - Object access violation (A mplid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (tmout is -2 or less, blksz is negative or invalid)

`E_DLT` - The object being waited for was deleted (the specified memorypool was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state; implementation dependent for pget_blk and tget_blk(tmout=TMO_POL))

DESCRIPTION:

A memory block of the size in bytes given by blksz is allocated from the variable-size memorypool specified by mplid. The start address of the memory block allocated is returned in blk. The allocated memory block is not cleared to zero. The contents of the memory block allocated are undefined. If the memory block cannot be obtained from the specified memorypool when get_blk is issued, the task issuing get_blk will be placed on the memory allocation queue of the specified memorypool, and wait until it can get the memory it requires.

The order in which tasks wait on the queue when waiting to be allocated memory blocks is according to either FIFO or task priority. The specification whereby tasks wait according to task priority is considered an extended function [level X] for which compatibility is not guaranteed. Furthermore, when tasks form a memory allocation queue, it is implementation dependent whether priority is given to tasks requesting the smaller size of memory or those at the head of the queue.

NOTES:

Pget_blk and get_blk represent the same processing as specifying certain values (TMO_POL or TMO_FEVR) to tget_blk for tmout. It is allowed that only tget_blk is implemented in the kernel and that pget_blk and get_blk are implemented as macros which call tget_blk.

9.4.4 pget_blk - Poll and Get Variable-Size Memory Block

CALLING SEQUENCE:

```
ER ercd =pget_blk(
  VP *p_blk,
  ID mplid,
  INT blksz
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mplid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mplid does not exist)

`E_OACV` - Object access violation (A mplid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (tmout is -2 or less, blksz is negative or invalid)

`E_DLT` - The object being waited for was deleted (the specified memorypool was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state; implementation dependent for pget_blk and tget_blk(tmout=TMO_POL))

DESCRIPTION:

The pget_blk system call has the same function as get_blk except for the waiting feature. Pget_blk polls whether or not the task should wait if get_blk is executed. The meaning of parameters to pget_blk are the same with get_blk. The specific operations by pget_blk are as follows.

- If there is enough free memory to get the memory block of specified size immediately, processing is the same as get_blk: that is, the requested memory is allocated and the system call completes normally.

- If there is not enough free memory, an E_TMOUT error is returned to indicate polling failed and the system call finishes. Unlike get_blk, the issuing task does not wait in this case. Also, the issuing task does not get any memory.

NOTES:

NONE

9.4.5 tget_blk - Get Variable-Size Memory Block with Timeout

CALLING SEQUENCE:

```
ER ercd =tget_blk(
  VP *p_blk,
  ID mplid,
  INT blksz,
  TMO tmout
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mplid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mplid does not exist)

`E_OACV` - Object access violation (A mplid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (tmout is -2 or less, blksz is negative or invalid)

`E_DLT` - The object being waited for was deleted (the specified memorypool was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state; implementation dependent for pget_blk and tget_blk(tmout=TMO_POL))

DESCRIPTION:

The tget_blk system call has the same function as get_blk with an additional timeout feature. A maximum wait time (timeout value) can be specified using the parameter tmout. When a timeout is specified, a timeout error, E_TMOUT, will result and the system call will finish if the period specified by tmout elapses without conditions for releasing wait being satisfied (i.e. without sufficient free memory becoming available).

NOTES:

NONE

9.4.6 rel_blk - Release Variable-Size Memory Block

CALLING SEQUENCE:

```
ER rel_blk(
  ID mplid,
  VP blk
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mplid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mplid does not exist)

`E_OACV` - Object access violation (A mplid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (blk is invalid or an attempt was made to return the memory block to the wrong memorypool)

DESCRIPTION:

This system call releases the memory block specified by blk and returns it to the variable-size memorypool specified by mplid. Executing rel_blk allows memory to be allocated to the next task waiting for memory allocation from the memorypool given by mplid, thus releasing that task from its WAIT state. The variable-size memorypool to which the memory block is returned must be the same memorypool from which it was originally allocated. An E_PAR error will result if an attempt is made to return a memory block to another memorypool than that from which it was originally allocated.

NOTES:

When memory is returned to a variable-size memorypool for which more than one task is waiting, multiple tasks may be released from waiting at the same time depending on the number of bytes of memory. The order of tasks on the ready queue among tasks having the same priority after being released from waiting for memory is implementation dependent.

9.4.7 ref_mpl - Reference Variable-Size Memorypool Status

CALLING SEQUENCE:

```
ER ref_mpl(
  T_RMPL *pk_rmpl,
  ID mplid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mplid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mplid does not exist)

`E_OACV` - Object access violation (A mplid less than -4 was specified from a user task. This is implementation dependent.) Note: User tasks can issue ref_mpl in order to reference memorypools of mplid = TMPL_OS = -4. Whether or not memorypools of mplid = -3 or -2 may be specified to ref_mpl by user tasks is implementation dependent.

`E_PAR` - Parameter error (the packet address for the return parameters could not be used)

DESCRIPTION:

This system call refers to the state of the variable-size memorypool specified by mplid, and returns the total free memory size currently available (frsz), the maximum continuous memory size of readily available free memory (maxsz), information of a task waiting to be allocated memory (wtsk), and its extended information (exinf). Wtsk indicates, whether or not there is a task waiting to be allocated memory from the variable-size memorypool specified. If there is no waiting task, wtsk is returned as FALSE = 0. If there is a waiting task, wtsk is returned as a value other than 0.

NOTES:

In some implementations, memorypools having mplid = -3 or -2 may be referred with ref_mpl as memorypools used by implementation-dependent parts of the OS (such as those used for the stack only). This system call can provide more detailed information regarding remaining memory if the usage of memorypools having mplid = -3 or -2 is more clearly defined. Whether or not this feature is used and any details regarding information provided are implementation dependent.

10 Fixed Block Manager

10.1 Introduction

The fixed block manager provides functions for creating, deleting, getting, polling, getting with timeout, releasing, and referencing the fixed-sized memorypool. This manager is based on ITRON 3.0 standard.

The services provided by the fixed block manager are:

- `cre_mpf` - Create Fixed-Size Memorypool
- `del_mpf` - Delete Fixed-Size Memorypool
- `get_blf` - Get Fixed-Size Memory Block
- `pget_blf` - Poll and Get Fixed-Size Memory Block
- `tget_blf` - Get Fixed-Size Memory Block with Timeout
- `rel_blf` - Release Fixed-Size Memory Block
- `ref_mpf` - Reference Fixed-Size Memorypool Status

10.2 Background

10.3 Operations

10.4 System Calls

This section details the fixed block manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

10.4.1 cre_mpf - Create Fixed-Size Memorypool

CALLING SEQUENCE:

```
ER cre_mpf(
  ID mpfid,
  T_CMPF *pk_cmpf
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_NOMEM` - Insufficient memory (Memory for control block and/or for memorypool cannot be allocated)

`E_ID` - Invalid ID number (mpfid was invalid or could not be used)

`E_RSATR` - Reserved attribute (mpfatr was invalid or could not be used)

`E_OBJ` - Invalid object state (a memorypool of the same ID already exists)

`E_OACV` - Object access violation (A mpfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (pk_cmpf is invalid or mpfsz and/or blfsz is negative or invalid)

DESCRIPTION:

This system call creates a fixed-size memorypool having the ID number specified by mpfid. Specifically, a memory area of a size based on values of mpfcnt and blfsz is reserved for use as a memorypool. A control block for the memorypool being created is also allocated. The get_blf system call specifying the memorypool created by this call can be issued to allocate memory blocks of the size given by blfsz (in bytes).

NOTES:

The memory area required for creating memorypools and for allocating control blocks for each object is allocated while system initialization. Associated parameters are therefore specified at system configuration.

10.4.2 del_mpf - Delete Fixed-Size Memorypool

CALLING SEQUENCE:

```
ER del_mpf(
  ID mpfid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mpfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mpfid does not exist)

`E_OACV` - Object access violation (A mpfid less than -4 was specified from a user task. This is implementation dependent.)

DESCRIPTION:

This system call deletes the fixed-size memorypool specified by mpfid. No check or error report is performed even if there are tasks using memory from the memorypool to be deleted. This system call completes normally even if some of the memory blocks are not returned. Issuing this system call causes memory used for the control block of the associated memorypool and the memory area making up the memorypool itself to be released. After this system call is invoked, another memorypool having the same ID number can be created. This system call will complete normally even if there are tasks waiting to get memory blocks from the memorypool. In that case, an E_DLT error will be returned to each waiting task.

NOTES:

When a memorypool being waited for by more than one tasks is deleted, the order of tasks on the ready queue after the WAIT state is cleared is implementation dependent in the case of tasks having the same priority.

10.4.3 get_blf - Get Fixed-Size Memory Block

CALLING SEQUENCE:

```
ER get_blf(
  VP *p_blf,
  ID mpfid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mpfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mpfid does not exist)

`E_OACV` - Object access violation (A mpfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the specified memorypool was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state; implementation dependent for pget_blf and tget_blf(tmout=TMO_POL))

DESCRIPTION:

A memory block is allocated from the fixed-size memorypool specified by mpfid. The start address of the memory block allocated is returned to blf. The size of the memory block allocated is specified by the blfsz parameter when the fixed-size memorypool was created. The allocated memory block is not cleared to zero. The contents of the allocated memory block are undefined. If the memory block cannot be obtained from the specified memorypool when get_blf is issued, the task issuing get_blf will be placed on the memory allocation queue of the specified memorypool, and wait until it can get the memory it requires. If the object being waited for is deleted (the specified memorypool is deleted while waiting), an E_DLT error will be returned.

NOTES:

NONE

10.4.4 pget_blf - Poll and Get Fixed-Size Memory Block

CALLING SEQUENCE:

```
ER ercd =pget_blf(
  VP *p_blf,
  ID mpfid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mpfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mpfid does not exist)

`E_OACV` - Object access violation (A mpfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the specified memorypool was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state; implementation dependent for pget_blf and tget_blf(tmout=TMO_POL))

DESCRIPTION:

The pget_blf system call has the same function as get_blf except for the waiting feature. Pget_blf polls whether or not the task should wait if get_blf is executed. The meaning of parameters to pget_blf are the same with get_blf. The specific operations by pget_blf are as follows.

- If there is a free memory block available, processing is the same as get_blf: that is, the requested memory is allocated and the system call completes normally.

- If there is no free memory block, an E_TMOUT error is returned to indicate polling failed and the system call finishes. Unlike get_blf, the issuing task does not wait in this case. Also, the issuing task does not get any memory.

NOTES:

NONE

10.4.5 tget_blf - Get Fixed-Size Memory Block with Timeout

CALLING SEQUENCE:

```
ER ercd =tget_blf(
  VP *p_blf,
  ID mpfid,
  TMO tmout
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mpfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mpfid does not exist)

`E_OACV` - Object access violation (A mpfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (tmout is -2 or less)

`E_DLT` - The object being waited for was deleted (the specified memorypool was deleted while waiting)

`E_RLWAI` - WAIT state was forcibly released (rel_wai was received while waiting)

`E_TMOUT` - Polling failure or timeout exceeded

`E_CTX` - Context error (issued from task-independent portions or a task in dispatch disabled state; implementation dependent for pget_blf and tget_blf(tmout=TMO_POL))

DESCRIPTION:

The tget_blf system call has the same function as get_blf with an additional timeout feature. A maximum wait time (timeout value) can be specified using the parameter tmout. When a timeout is specified, a timeout error, E_TMOUT, will result and the system call will finish if the period specified by tmout elapses without conditions for releasing wait being satisfied (i.e. without free memory becoming available).

NOTES:

NONE

10.4.6 rel_blf - Release Fixed-Size Memory Block

CALLING SEQUENCE:

```
ER rel_blf(
  ID mpfid,
  VP blf
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mpfid was invalid or could not be used)

`E_NOEXS` - Object does not exist (the memorypool specified by mpfid does not exist)

`E_OACV` - Object access violation (A mpfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (blf is invalid or an attempt was made to return the memory block to the wrong memorypool)

DESCRIPTION:

This system call releases the memory block specified by blf and returns it to the fixed-size memorypool specified by mpfid. Executing rel_blf allows memory to be allocated to the next task waiting for memory allocation from the memorypool given by mpfid, thus releasing that task from its WAIT state.

NOTES:

The fixed-size memorypool to which the memory block is returned must be the same memorypool from which it was originally allocated. An E_PAR error will result if an attempt is made to return a memory block to another memorypool than that from which it was originally allocated.

10.4.7 ref_mpf - Reference Fixed-Size Memorypool Status

CALLING SEQUENCE:

```
ER ref_mpf(
  T_RMPF *pk_rmpf,
  ID mpfid
);
```

STATUS CODES:

`E_OK` - Normal Completion

`E_ID` - Invalid ID number (mpfid was invalid or could not be used)

`E_NOEXS` - Object does not exist \(the memorypool specified by mpfid does not exist.)

`E_OACV` - Object access violation (A mpfid less than -4 was specified from a user task. This is implementation dependent.)

`E_PAR` - Parameter error (the packet address for the return parameters could not be used)

DESCRIPTION:

This system call refers to the state of the fixed-size memorypool specified by mpfid, and returns the current number of free blocks (frbcnt), information of a task waiting to be allocated memory (wtsk), and its extended information (exinf). Wtsk indicates whether or not there is a task waiting to be allocated memory from the fixed-size memorypool specified. If there is no waiting task, wtsk is returned as FALSE = 0. If there is a waiting task, wtsk is returned as a value other than 0.

NOTES:

While the frsz return parameter of ref_mpl returns the total size of free memory, the frbcnt return parameter of ref_mpf returns the number of free blocks.

Depending on the implementation, additional information besides wtsk and frbcnt (such as memorypool attributes, mpfatr) may also be referred.

11 Time Manager

11.1 Introduction

The time manager is ...

The services provided by the time manager are:

- `get_tim` - Get System Clock
- `set_tim` - Set System Clock
- `dly_tsk` - Delay Task
- `def_cyc` - Define Cyclic Handler
- `act_cyc` - Activate Cyclic Handler
- `ref_cyc` - Reference Cyclic Handler Status
- `def_alm` - Define Alarm Handler
- `ref_alm` - Reference Alarm Handler Status
- `ret_tmr` - Return from Timer Handler

11.2 Background

11.3 Operations

11.4 System Calls

This section details the time manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

11.4.1 get_tim - Get System Clock

CALLING SEQUENCE:

```
ER get_tim(
  SYSTIME *pk_tim
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

11.4.2 set_tim - Set System Clock

CALLING SEQUENCE:

```
ER set_tim(
  SYSTIME *pk_tim
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

11.4.3 dly_tsk - Delay Task

CALLING SEQUENCE:

```
ER dly_tsk(
  DLYTIME dlytim
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

11.4.4 def_cyc - Define Cyclic Handler

CALLING SEQUENCE:

```
ER def_cyc(
  HNO cycno,
  T_DCYC *pk_dcyc
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

11.4.5 act_cyc - Activate Cyclic Handler

CALLING SEQUENCE:

```
ER act_cyc(
  HNO cycno,
  UINT cycact
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

11.4.6 ref_cyc - Reference Cyclic Handler Status

CALLING SEQUENCE:

```
ER ref_cyc(
  T_RCYC *pk_rcyc,
  HNO cycno
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

11.4.7 def_alm - Define Alarm Handler

CALLING SEQUENCE:

```
ER def_alm(
  HNO almno,
  T_DALM *pk_dalm
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

11.4.8 ref_alm - Reference Alarm Handler Status

CALLING SEQUENCE:

```
ER ref_alm(
  T_RALM *pk_ralm,
  HNO almno
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

11.4.9 ret_tmr - Return from Timer Handler

CALLING SEQUENCE:

```
void ret_tmr(

);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

12 System Manager

12.1 Introduction

The system manager is ...

The services provided by the system manager are:

- `get_ver` - Get Version Information
- `ref_sys` - Reference Semaphore Status
- `ref_cfg` - Reference Configuration Information
- `def_svc` - Define Extended SVC Handler
- `def_exc` - Define Exception Handler

12.2 Background

12.3 Operations

12.4 System Calls

This section details the system manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

12.4.1 get_ver - Get Version Information

CALLING SEQUENCE:

```
ER get_ver(
 T_VER *pk_ver
);
```

STATUS CODES:

`EXXX` -

DESCRIPTION:

NOTES:

12.4.2 ref_sys - Reference Semaphore Status

CALLING SEQUENCE:

```
ER ref_sys(
 T_RSYS *pk_rsys
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

12.4.3 ref_cfg - Reference Configuration Information

CALLING SEQUENCE:

```
ER ref_cfg(
 T_RCFG *pk_rcfg
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

12.4.4 def_svc - Define Extended SVC Handler

CALLING SEQUENCE:

```
ER def_svc(
  FN s_fncd,
  T_DSVC *pk_dsvc
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

12.4.5 def_exc - Define Exception Handler

CALLING SEQUENCE:

```
ER def_exc(
  UINT exckind,
  T_DEXC *pk_dexc
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

13 Network Support Manager

13.1 Introduction

The network support manager is ...

The services provided by the network support manager are:

- `nrea_dat` - Read Data from another Node
- `nwri_dat` - Write Data to another Node
- `nget_nod` - Get Local Node Number
- `nget_ver` - Get Version Information of another Node

13.2 Background

13.3 Operations

13.4 System Calls

This section details the network support manager's services. A subsection is dedicated to each of this manager's services and describes the calling sequence, related constants, usage, and status codes.

13.4.1 nrea_dat - Read Data from another Node

CALLING SEQUENCE:

```
ER nrea_dat(
  INT *p_reasz,
  VP dstadr,
  NODE srcnode,
  VP srcadr,
);
```

STATUS CODES:

`EXXX` -

DESCRIPTION:

NOTES:

13.4.2 nwri_dat - Write Data to another Node

CALLING SEQUENCE:

```
ER nwri_dat(
  INT *p_wrisz,
  NODE dstnode,
  VP dstadr,
  VP srcadr,
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

13.4.3 nget_nod - Get Local Node Number

CALLING SEQUENCE:

```
ER nget_nod(
  NODE *p_node
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

13.4.4 nget_ver - Get Version Information of another Node

CALLING SEQUENCE:

```
ER nget_ver(
  T_VER *pk_ver,
  NODE node
);
```

STATUS CODES:

EXXX -

DESCRIPTION:

NOTES:

14 ITRON Implementation Status

14.1 Introduction

This chapter describes the status of the implementation of each manager in the RTEMS implementataion of the uITRON 3.0 API. The status of each manager is presented in terms of documentation and status relative to the extended level (level 'E') of the uITRON 3.0 API specification. The extended level of the specification is the level at which dynamic object creation, deletion, and reference services are available. This level is more akin to the other APIs supported by RTEMS. This purpose of this chapter is to make it clear what is required to bring the RTEMS uITRON API implementation into compliance with the specification. The following description of the specification levels is taken from the uITRON 3.0 API specification.

uITRON 3.0 specification is divided into fewer system call levels than was the previous uITRON 2.0 specification. There are now just three levels: Level R (Required), Level S (Standard) and Level E (Extended). In addition to these three levels, there is also Level C for CPU-dependent system calls. In addition, the uITRON 3.0 API, defines the network level ('N') which represents system calls that support the connection function

- [level R] (Required) The functions in this level are mandatory for all implementations of uITRON 3.0 specification. This includes basic functions for achieving a real-time, multitasking OS. These functions can be implemented even without a hardware timer. This level corresponds to Levels 1 and 2 of uITRON 2.0 specification.
- [level S] (Standard) This includes basic functions for achieving a real-time, multitasking OS. This level corresponds to Levels 3 and 4 of uITRON 2.0 specification.
- [level E] (Extended) This includes additional and extended functions. This corresponds to functions not included in uITRON 2.0 specification (functions of ITRON2 specification). Specifically, this level includes object creation and deletion functions, rendezvous functions, memorypools and the timer handler.
- [level C] (CPU dependent) This level provides implementation-dependent functions required due to the CPU or hardware configuration.

The support level of the connection function is indicated by appending an 'N' to the end of the level. For example, connectivity supported at [level S] would be referred to as [level SN]. The support level for functions which can only send requests for operations on other nodes but offer no system call processing on the issuing node itself are indicated by the lower case letter 's' or 'e'.

14.2 Task Status

- Implementation
    - cre_tsk - Complete, Pending Review
    - del_tsk - Complete, Pending Review
    - sta_tsk - Complete, Pending Review
    - ext_tsk - Complete, Pending Review
    - exd_tsk - Complete, Pending Review

    - ter_tsk - Complete, Pending Review
    - dis_dsp - Complete, Pending Review
    - ena_dsp - Complete, Pending Review
    - chg_pri - Complete, Pending Review
    - rot_rdq - Complete, Pending Review
    - rel_wai - Stub, Needs to be Fleshed Out
    - get_tid - Complete, Pending Review
    - ref_tsk - Complete, Pending Review
    - Notes:
        - None
- Executive Modifications
    - None Expected
- Testing
    - itron01 - Hello world
    - itron02 - Semaphore test
    - itron03 - directives: ex_init, ex_start, t_create, t_start, tm_tick, i_return, t_ident, tm_set, tm_get, tm_wkafter See .doc file, verify correct
    - itron04 - Doc file needed
    - itron05 - directives: ext_tsk, cre_tsk, sta_tsk, rot_rdq ex_start, t_create, t_start, tm_tick, i_return, t_ident, t_delete, tm_wkafter, t_setpri, t_suspend See .doc file, verify correct
    - itron06 - Doc file needed
    - itron07 - Doc file needed
    - itron08 - Doc file needed
    - itron09 - Doc file needed
    - itron10 - Doc file needed
    - tmitron01 - Doc file needed
    - tm_include - Doc file needed. Timing test for semaphores.
- Documentation
    - Complete, Pending Review
- ITRON 3.0 API Conformance
    - Level E - Extended Functionality
- Level C - CPU Dependent Functionality
    - NA
- Level N - Connection Functionality
    - Not implemented

14.3 Task-Dependent Synchronization Status

- Implementation
  - sus_tsk - Complete, Pending Review
  - rsm_tsk - Complete, Pending Review
  - frsm_tsk - Complete, Pending Review
  - slp_tsk - Stub, Needs to be Fleshed Out
  - tslp_tsk - Stub, Needs to be Fleshed Out
  - wup_tsk - Stub, Needs to be Fleshed Out
  - can_wup - Stub, Needs to be Fleshed Out
  - Notes:
    - None
- Executive Modifications
  - None Expected
- Testing
  - Functional tests for complete routines.
  - Yellow line testing needs to be verified.
  - No Timing Tests
- Documentation
  - Complete, Pending Review
  -
- ITRON 3.0 API Conformance
  - Level E - Extended Functionality
  - Level C - CPU Dependent Functionality
    - NA
  - Level N - Connection Functionality
    - Not implemented

14.4 Semaphore

- Implementation
  - cre_sem - Complete, Pending Review
  - del_sem - Complete, Pending Review
  - sig_sem - Complete, Pending Review
  - wai_sem - Complete, Pending Review
  - preq_sem - Complete, Pending Review
  - twai_sem - Complete, Pending Review
  - ref_sem - Complete, Pending Review
- Executive Modifications
  - None Required

- Testing
  - Yellow Lined BUT Timeout Cases Not Actually Executed
  - No Timing Tests
- Documentation
  - Complete, Pending Review
- ITRON 3.0 API Conformance
  - Level E - Extended Functionality
    - Complete, Pending Review
  - Level C - CPU Dependent Functionality
    - NA
  - Level N - Connection Functionality
    - Not implemented

14.5 Eventflags

- Implementation
  - cre_flg - Stub, Needs to be Fleshed Out
  - del_flg - Stub, Needs to be Fleshed Out
  - set_flg - Stub, Needs to be Fleshed Out
  - clr_flg - Stub, Needs to be Fleshed Out
  - wai_flg - Stub, Needs to be Fleshed Out
  - pol_flg - Stub, Needs to be Fleshed Out
  - twai_flg - Stub, Needs to be Fleshed Out
  - ref_flg - Stub, Needs to be Fleshed Out
  - Notes:
    - Similar in Functionality to Classic API Events Manager
    - Implement Using new SuperCore Event Handler
- Executive Modifications
  - Add SuperCore Events Object Based on Classic Events
  - Redo Classic Events to use SuperCore Events
- Testing
  - No Tests Written
  - No Timing Tests
- Documentation
  - Good First Draft.
  - No Information in Operations.
  - Should not use "standard-like" language.
- ITRON 3.0 API Conformance
  - Level E - Extended Functionality
    -

- Level C - CPU Dependent Functionality
  - NA
- Level N - Connection Functionality
  - Not implemented

14.6 Mailbox

- Implementation
  - cre_mbx - Stub, Needs to be Fleshed Out
  - del_mbx - Stub, Needs to be Fleshed Out
  - snd_msg - Stub, Needs to be Fleshed Out
  - rcv_msg - Stub, Needs to be Fleshed Out
  - prcv_msg - Stub, Needs to be Fleshed Out
  - trcv_msg - Stub, Needs to be Fleshed Out
  - ref_mbx - Stub, Needs to be Fleshed Out
  - Notes:
    - Passes Addresses of Messages
    - FIFO or Priority Task Blocking
    - FIFO or Priority Message Ordering
    - Send Returns Error on Overflow
- Executive Modifications
  - None
- Testing
  - No Tests Written
  - No Timing Tests
- Documentation
  - Needs More Text
  - No Information in Background or Operations.
  - Service Descriptions are Weak.
- ITRON 3.0 API Conformance
  - Level E - Extended Functionality
    -
  - Level C - CPU Dependent Functionality
    - NA
  - Level N - Connection Functionality
    - Not implemented

14.7 Message Buffer

- Implementation
  - cre_mbf - Stub, Needs to be Fleshed Out
  - del_mbf - Stub, Needs to be Fleshed Out
  - snd_mbf - Stub, Needs to be Fleshed Out
  - psnd_mbf - Stub, Needs to be Fleshed Out
  - tsnd_mbf - Stub, Needs to be Fleshed Out
  - rcv_mbf - Stub, Needs to be Fleshed Out
  - prcv_mbf - Stub, Needs to be Fleshed Out
  - trcv_mbf - Stub, Needs to be Fleshed Out
  - ref_mbf - Stub, Needs to be Fleshed Out
  - Notes:
    - Implement Using SuperCore Message Queue Handler
    - Passes Full Bodies of Messages
    - FIFO or Priority Task Blocking
    - FIFO Message Ordering Only
    - Send (snd_mbf and tsnd_mbf) Blocks on Overflow
- Executive Modifications
  - SuperCore Message Queue Handler Must Support Blocking Sends. [NOTE: This is required for POSIX Message Queues as well.]
- Testing
  - No Tests Written
  - No Timing Tests
- Documentation
  - Good First Draft.
  - No Information in Operations
- ITRON 3.0 API Conformance
  - Level E - Extended Functionality
    -
  - Level C - CPU Dependent Functionality
    - NA
  - Level N - Connection Functionality
    - Not implemented

14.8 Rendezvous

- Implementation
  - cre_por - Stub, Needs to be Fleshed Out
  - del_por - Stub, Needs to be Fleshed Out
  - cal_por - Stub, Needs to be Fleshed Out

- pcal_por - Stub, Needs to be Fleshed Out
- tcal_por - Stub, Needs to be Fleshed Out
- acp_por - Stub, Needs to be Fleshed Out
- pacp_por - Stub, Needs to be Fleshed Out
- tacp_por - Stub, Needs to be Fleshed Out
- fwd_por - Stub, Needs to be Fleshed Out
- rpl_rdv - Stub, Needs to be Fleshed Out
- ref_por - Stub, Needs to be Fleshed Out
- Notes:
  - Hardest ITRON Manager to Implement

- Executive Modifications
  - Doubtful, Probably Implement in Terms of Multiple SuperCore Objects.
- Testing
  - No Tests Written
  - No Timing Tests
- Documentation
  - Shell, Needs to be Fleshed Out
- ITRON 3.0 API Conformance
  - Level E - Extended Functionality
    -
  - Level C - CPU Dependent Functionality
    - NA
  - Level N - Connection Functionality
    - Not implemented

14.9 Interrupt

- Implementation
  - def_int - Stub, Needs to be Fleshed Out
  - ret_int - Stub, Needs to be Fleshed Out
  - ret_wup - Stub, Needs to be Fleshed Out
  - loc_cpu - Stub, Needs to be Fleshed Out
  - unl_cpu - Stub, Needs to be Fleshed Out
  - dis_int - Stub, Needs to be Fleshed Out
  - ena_int - Stub, Needs to be Fleshed Out
  - chg_iXX - Stub, Needs to be Fleshed Out
  - ref_iXX - Stub, Needs to be Fleshed Out
  - Notes:

        - This quote from the ITRON specification needs to be thought about:

          "*When an interrupt is invoked, the interrupt handler defined with this system call is started directly by the interrupt processing mechanism of the CPU hardware. Accordingly, code at the beginning and end of an interrupt handler must save and restore any registers used by the interrupt handler.*"

          Based on another comment, in the ret_int description, I think this means that RTEMS will not support the TA_ASM style of interrupt handlers – only the TA_HLNG style.

          When TA_HLNG is specified, a high-level language environment setting program (a high-level language support routine) is called before branching to the inthdr address. The least significant bit (LSB) of the system attribute bits is used for this specification.
        - Specification allows special "interrupt-only" versions of system calls named i???_??? (i.e. sig_sem and isig_sem). This does not seem to be something that would be implemented with RTEMS. We could provide macros mapping them onto the default versions if this is an issue.
        - How this operates versus the behavior of a true TRON chip is up for discussion.
        - ret_wup is questionable in only high-level language ISRs.
        - dis_int and ena_int refer to a specific interrupt number. These may require hooking back out to the BSP.
        - for chg_iXX and reg_iXX, the XX should be replaced with something that is meaningful on a particular CPU.
- Executive Modifications
    - None Expected
- Testing
    - No Tests Written
    - No Timing Tests
- Documentation
    - Shell, Needs to be Fleshed Out
- ITRON 3.0 API Conformance
    - Level E - Extended Functionality
        -
    - Level C - CPU Dependent Functionality
        - NA
    - Level N - Connection Functionality
        - Not implemented

14.10 Memory Pool

- Implementation
    - cre_mpl - Stub, Needs to be Fleshed Out
    - del_mpl - Stub, Needs to be Fleshed Out
    - get_blk - Stub, Needs to be Fleshed Out
    - pget_blk - Stub, Needs to be Fleshed Out
    - tget_blk - Stub, Needs to be Fleshed Out
    - rel_blk - Stub, Needs to be Fleshed Out
    - ref_mpl - Stub, Needs to be Fleshed Out
    - Notes:
        - Implement Using SuperCore Heap Handler
        - Similar to Region in Classic API with Blocking
        - FIFO or Priority Task Blocking
        - Specification Deliberately Open on Allocation Algorithm
        - Multiple Tasks Can be Unblocked by a single rel_blk
- Executive Modifications
    - None Expected
- Testing
    - No Tests Written
    - No Timing Tests
- Documentation
    - Good First Draft.
    - No Information in Operations
    - Should not use "standard-like" language.
- ITRON 3.0 API Conformance
    - Level E - Extended Functionality
        -
    - Level C - CPU Dependent Functionality
        - NA
    - Level N - Connection Functionality
        - Not implemented

14.11 Fixed Block

- Implementation
    - cre_mpf - Stub, Needs to be Fleshed Out
    - del_mpf - Stub, Needs to be Fleshed Out
    - get_blf - Stub, Needs to be Fleshed Out
    - pget_blf - Stub, Needs to be Fleshed Out

- tget_blf - Stub, Needs to be Fleshed Out
- rel_blf - Stub, Needs to be Fleshed Out
- ref_mpf - Stub, Needs to be Fleshed Out
- Notes:
  - Implement Using SuperCore Chain Handler
  - Similar to Partition in Classic API with Blocking
  - FIFO or Priority Task Blocking
  - Specification Deliberately Open on Allocation Algorithm
  - Should add Blocking to Classic API Partition at Same Time

- Executive Modifications
  - None Expected
- Testing
  - No Tests Written
  - No Timing Tests
- Documentation
  - Good First Draft.
  - No Information in Background or Operations
  - Should not use "standard-like" language.
- ITRON 3.0 API Conformance
  - Level E - Extended Functionality
    -
  - Level C - CPU Dependent Functionality
    - NA
  - Level N - Connection Functionality
    - Not implemented

14.12 Time

- Implementation
  - get_tim - Stub, Needs to be Fleshed Out
  - set_tim - Stub, Needs to be Fleshed Out
  - dly_tsk - Stub, Needs to be Fleshed Out
  - def_cyc - Stub, Needs to be Fleshed Out
  - act_cyc - Stub, Needs to be Fleshed Out
  - ref_cyc - Stub, Needs to be Fleshed Out
  - def_alm - Stub, Needs to be Fleshed Out
  - ref_alm - Stub, Needs to be Fleshed Out
  - ret_tmr - Stub, Needs to be Fleshed Out
  - Notes:
    - Need to Implement Time Conversion Routines

        - Epoch is January 1, 1985, 00:00:00 am (GMT).
        - Cyclic and Alarm Handlers may be TA_ASM or TA_HLNG.
        - Alarms may be Absolute or Relative Time based.
        - May Want to Implement a Timer Server Task
        - Termination via ret_tmr is Not Consistent with Current RTEMS Timers.
- Executive Modifications
    - None Expected
- Testing
    - No Tests Written
    - No Timing Tests
- Documentation
    - Have Version in Word
- ITRON 3.0 API Conformance
    - Level E - Extended Functionality
        -
    - Level C - CPU Dependent Functionality
        - NA
    - Level N - Connection Functionality
        - Not implemented

14.13 System

- Implementation
    - get_ver - Stub, Needs to be Fleshed Out
    - ref_sys - Stub, Needs to be Fleshed Out
    - ref_cfg - Stub, Needs to be Fleshed Out
    - def_svc - Stub, Needs to be Fleshed Out
    - def_exc - Stub, Needs to be Fleshed Out
    - Notes:
        - May Have to Obtain ITRON "OS Maker" Id
        - - def_svc seems to imply a trap handler interface
        - - def_exc needs to be examined.
- Executive Modifications
    - None Expected
- Testing
    - No Tests Written
    - No Timing Tests
- Documentation
    - Shell, Needs to be Fleshed Out

- ITRON 3.0 API Conformance
    - Level E - Extended Functionality
        -
    - Level C - CPU Dependent Functionality
        - NA
    - Level N - Connection Functionality
        - Not implemented

14.14 Network Support

- Implementation
    - nrea_dat - Stub, Needs to be Fleshed Out
    - nwri_dat - Stub, Needs to be Fleshed Out
    - nget_nod - Stub, Needs to be Fleshed Out
    - nget_ver - Stub, Needs to be Fleshed Out
    - Notes:
        - None of these are difficult to implement on top of MPCI
        - MP Packet formats are well-defined.
- Executive Modifications
    - None Expected
- Testing
    - No Tests Written
    - No Timing Tests
- Documentation
    - Shell, Needs to be Fleshed Out
- ITRON 3.0 API Conformance
    - Level E - Extended Functionality
        -
    - Level C - CPU Dependent Functionality
        - NA
    - Level N - Connection Functionality
        - Not implemented

Command and Variable Index

There are currently no Command and Variable Index entries.

Concept Index

There are currently no Concept Index entries.