µC3/Standard USERS GUIDE

3th Edition eForce Co., Ltd.

Introduction

uC3（Micro C cube） is a RTOS（Real Time Operating System）with kernel in accordance with specification of μITRON4.0, which is specified by TRON Association Corporation as Open Real Time kernel, in core.
The C3 of μC3 describes 3 concepts of **Compact , Connectivity, Capability.** Besides, the name of cube shows possibility of generating 3 effects by the concept above.

The position of this Document

This document is used as common manual for kernel function of μC3/Standard. However, another manual will be used for middle-ware or kernel function existing in each CPU. In case of necessity, please refer to these manuals.

TRON is abbreviation of "The Real-time Operation system Nucleus".
uITRON is abbreviation of "Micro Industrial TRON".
Specification of uITRON4.0 is available in homepage of TRON Association (http://www.assoc.tron.org/).
uC3 is a registered trademark of eForce Co., Ltd.
The contents of this document may be changed without prior notice.

Revision History

Modified items in 2nd edition

Page	Contents
	Change layout

Modified items in 3rd edition

Page	Description
17	In the explanation about System Service Block, corrected " that issued while an interrupt is processed" to "that invoked while an interrupt is processed ".
17	In the explanation about the error detection of system call in delayed execution, added “In order to detect the error has occurred during execution (except E_OK), the error handler is defined.”
24	Correct in Semaphore resource “Acquire・Release”
33	In the explanation of Memory pool management functions, added “and variable length memory pools” after “fixed-length memory pool”
35	Add explanation of updating time system and unit of time used in the kernel, timing that changed timeout time
43	Add the error handler as the peculiar function and its explanation
51,53,64,67, 75,93	At the explanation of the system calls, “in the error code of the return value”is added after「if this system call is invoked from the interrupt handler and has its execution delayed”
115	At the description of cre_mbf, corrected “maximum value of the implementation -defined”to「65535”
123	At the description of cre_por, corrected “maximum value of the implementation -defined”to「65535”
135	At the description of cre_mpf, deleted「if the value greater than the maximum value of the implementation-defined is specified”
195	Added “vdef_err” as the system call and its explanation
203	At the title of Stander COM port driver system call, corrected “receive one character”to “receive character string”
205,206	At the explanation of data types “TMO”「RELTIM”「SYSTIM”, corrected “unit of time is implementation-dependent”to “unit of time is 1ms”
201	At the explanation of data type「OVRTIM”, corrected「unit of time is implementation-dependent”to “unit of time is user-defined”

Table of Content

Chapter 1 What is μC3/Standard?

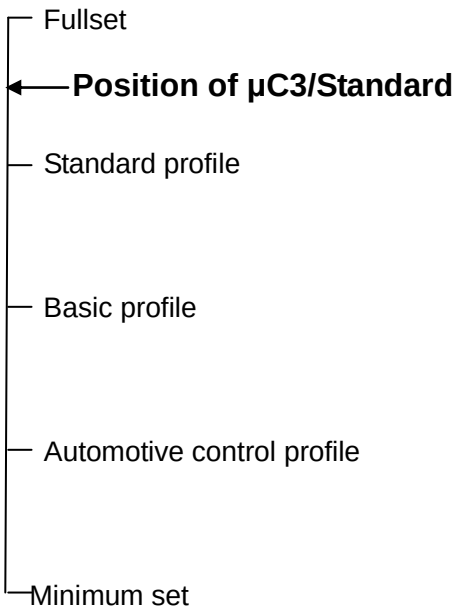
1.1 Features

μC3/Standard is a product focused in Capability in three concepts of μC3, include advanced and complex functions of μITRON without sacrificing the interrupt response.

Advanced and complex functions that might require a loop with no upper limit, but in μC3/Standard, there is not any loop can deteriorate the interrupt response during interrupt disabled period. Moreover, system calls and dispatchers are mostly executed in interrupt enabled state, so the interrupt response is being regarded as important.

1.2 Position in specification of μITRON

In specification of μITRON4.0,there are concept of profiles, but μC3/Standard does not match any profile, the functions that seem less likely to be used have been omitted based on fullset.



The functions omitted from fullset are as below:

- Static API and a configurator that interprets static API
- Task exception handling functions
- Service call management functions

Chapter 2 Basic concept of μC3/ Standard

2. 1 Glossary of basic terms

2. 1. 1 Task

A unit of a concurrent processing is called "Task". In other words, multiple tasks are executed concurrently when seen from an application's point of view. In fact, concurrent program that is the number of processor, a kernel make it seem like concurrent processing by using time-sharing techniques following the scheduling rules. The task that invokes a system call is called the "invoking task".

2. 1. 2 Dispatch and Scheduling

When processor switches the currently executing task, it is called "Dispatch", and mechanism of dispatch is "Dispatcher".

The act of selecting which task is to be executed next is called "Scheduling", and mechanism of scheduling is "Scheduler".

Generally, Dispatcher and Scheduler are hardly separated in definition. In uC3, they are integrated and called "Dispatcher" and "Dispatch".

2. 1. 3 Context

The environment for program execution is called "Context", and each task, time event handler or interrupt handler is considered to have its own context. In case of switching from one context to another, it is general to use context as a register value of processor because data necessary to restart must be saved and be retrieved.

2. 1. 4 Object and ID number

The resources on which a kernel or a software component operates are generally referred to as Object, the numbers which are used to identify and distinguish objects are called ID Number. In uC3/Standard, ID number is specified by configuration, beginning at 1 and rising to a maximum value. ID number of Object consist Object name + ID, such as Task ID, Semaphore ID.

Kernel objects include tasks, semaphores, event flags, data queues, mailboxes, mutexes, message buffers, rendezvous, fixed-length/variable-length memory pools, cyclic handlers, alarm handlers and interrupt service routines.

2. 1. 5 Service Call and System Call

The interface which invokes kernel or software component from application is called Service Call. In μC3, the Service Call of the kernel is called a system Call.

2. 1. 6 Priority order and priority level

The order relation decide the order of executing process, it is called “Priority order”, and parameter which is given by an application for that process execution is called “Priority Level”. Priority Level is represented by numeric value (natural number), a lower value indicates a higher precedence and vice versa.

In Priority Level of Task, there are Base Priority Level and Current Priority Level.

2. 1. 7 Preemptive

If a task which has priority higher than the running task becomes ready, be able to dispatched, it is called “Preemptive”

2. 1. 8 Time Tick

System time is managed in kernel. The event at constant period for counting system time is called 「Time Tick”. In other words, if a time tick cycle is 1mm second, accuracy of system time is 1mm second and if it is a cycle of 2mm second, its accuracy is 2mm second.

2. 1. 9 Queuing

The reservation function in case that there is some process requirement but it is not able to execute immediately is called “Queuing”, integrated as the counter to count the number of requires. In Queuing, there are activation request queuing and wakeup request queuing

2. 1. 10 Queue

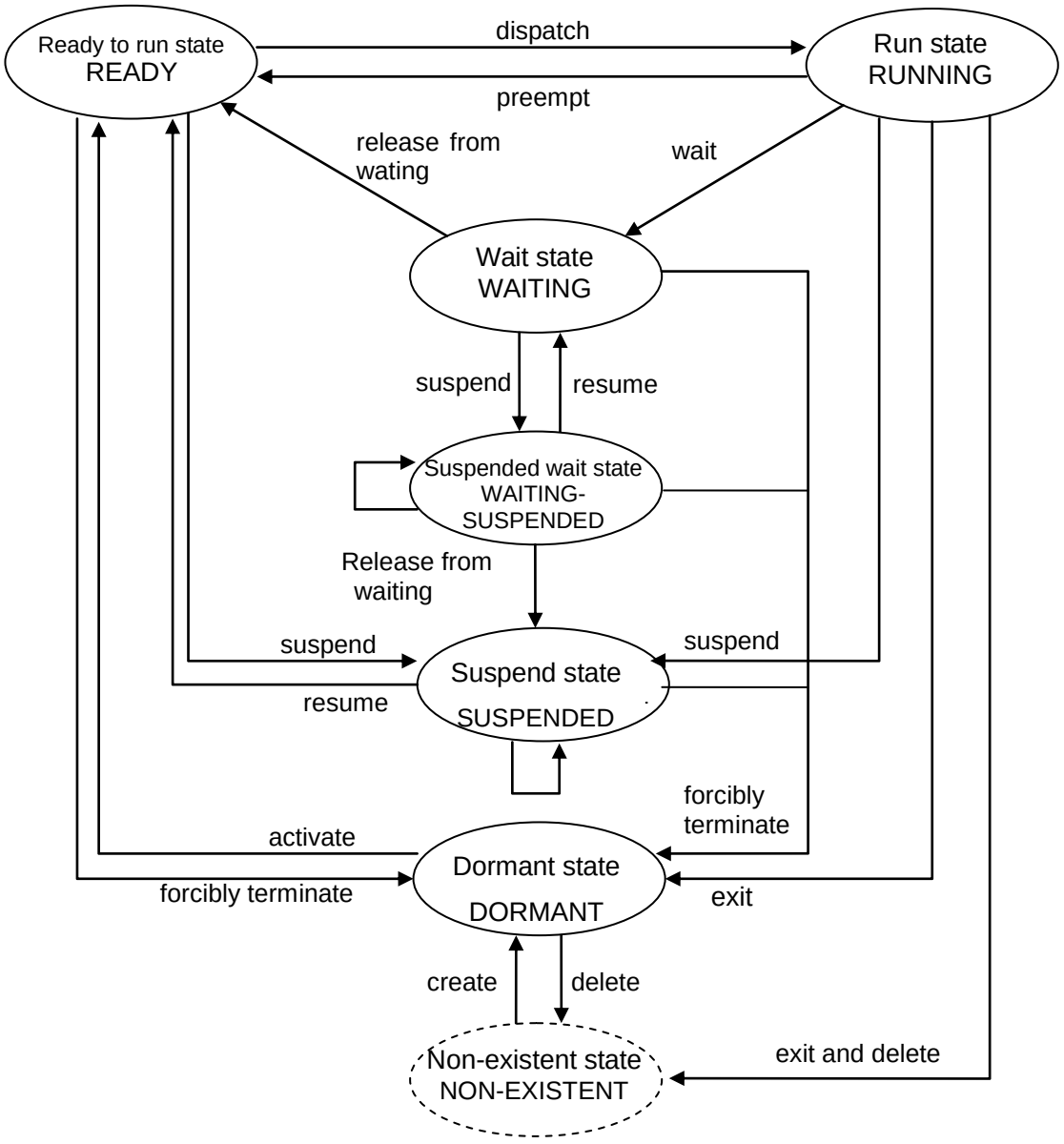
In case of invoking the system call to request a certain object, but the process is not able to be executed immediately, there will be system call which can wait till the process is able to execute or wait in a time limit. In the system call like this, it is queuing by the invoking of system call, and sequentially processed from the earliest one. This function is called “Queue”.

In queue, there also has the queue in order of task priority to process from higher priority task. In this case, if there are a number of tasks with the same priority, the first task added to the queue will be the first one to be handled.

2. 2 Task States and Scheduling Rule

2. 2. 1 Task States

In uC3/Standard, task states are classified into 7 broad categories, the blocked state category can be further broken down into three sub-states. The RUNNING state and the READY state are both generically reffered to as the “Runnable state“.
The following diagram illustrates the state transitions of tasks.



A Run state (RUNNING)

The task in this state is currently executing. Only one RUN state task exists at one time. Scheduler will decide one task among the tasks in the ready to run state, then Dispatcher moves the choosen task to the Run state. In other words, when non-task contexts take over execution,the task that was executing remains in the Run state.

B Ready to Run state（READY）

When a task is in the READY state,the task is ready to execute, but it cannot for some reason. In other words, there is a task with a higher priority is being executed or in state that dispatch dose not happen. “The state of dispatch does not happen“ in uC3 means that state of dispatch prohibition and interrupted mask are given precedence over task level.

C Blocked state

In a state of waiting for specific conditions to be met, the context of the task is stored in task management area to be able to resume.

The blocked state can be further classified into three sub-states.

C．1 Wait state（WAITING）

The task is blocked from executing because the conditions necessary for invoking system call have not yet been met. Concretely, wait state are classified as waiting for wakeup,waiting for fix time,waiting for event flags, waiting for semaphore, waiting for receiving a message from mailbox, waiting for transmission/reception a message of data queue,waiting for locking of mutex, waiting for transmission/reception from message buffers,waiting for memory block acquisition from fixed-length/variabe-length rendezvous, waiting for rendezvous call and waiting for acceptance/termination.

C．2 Suspend state（SUSPENDED）

This is the state where the task has been forcibly made to halt execution by another task or task itself.

C．3 Suspended wait state（WAITING-SUSPENDED）

This is the state where the task is both in WAITING state and SUSPENDED state. A task in the WAITING state goes to WAITING-SUSPENDED state if there is a request to move it to the SUSPENDED state.

D Dormant state（DORMANT）

This is the state where the tasks do not start or have already been terminated. The information in RUNNING state will not be saved while the task is in the DORMANT state. When a task is activated form the DORMANT state, it will begin executing from the task’s start

address.

E Non-existent state (NON-EXISTENT)

This is a virtual state where the task does not exists in the system, because it is not generated or has already been deleted.

Task activation means that a task in the DORMANT state is moved to the READY state, all the states other than the DORMANT state and NON-EXISTENT state are generically referred to as 「Active state “. Task termination means that a task in active state is moved to the DORMANT state.

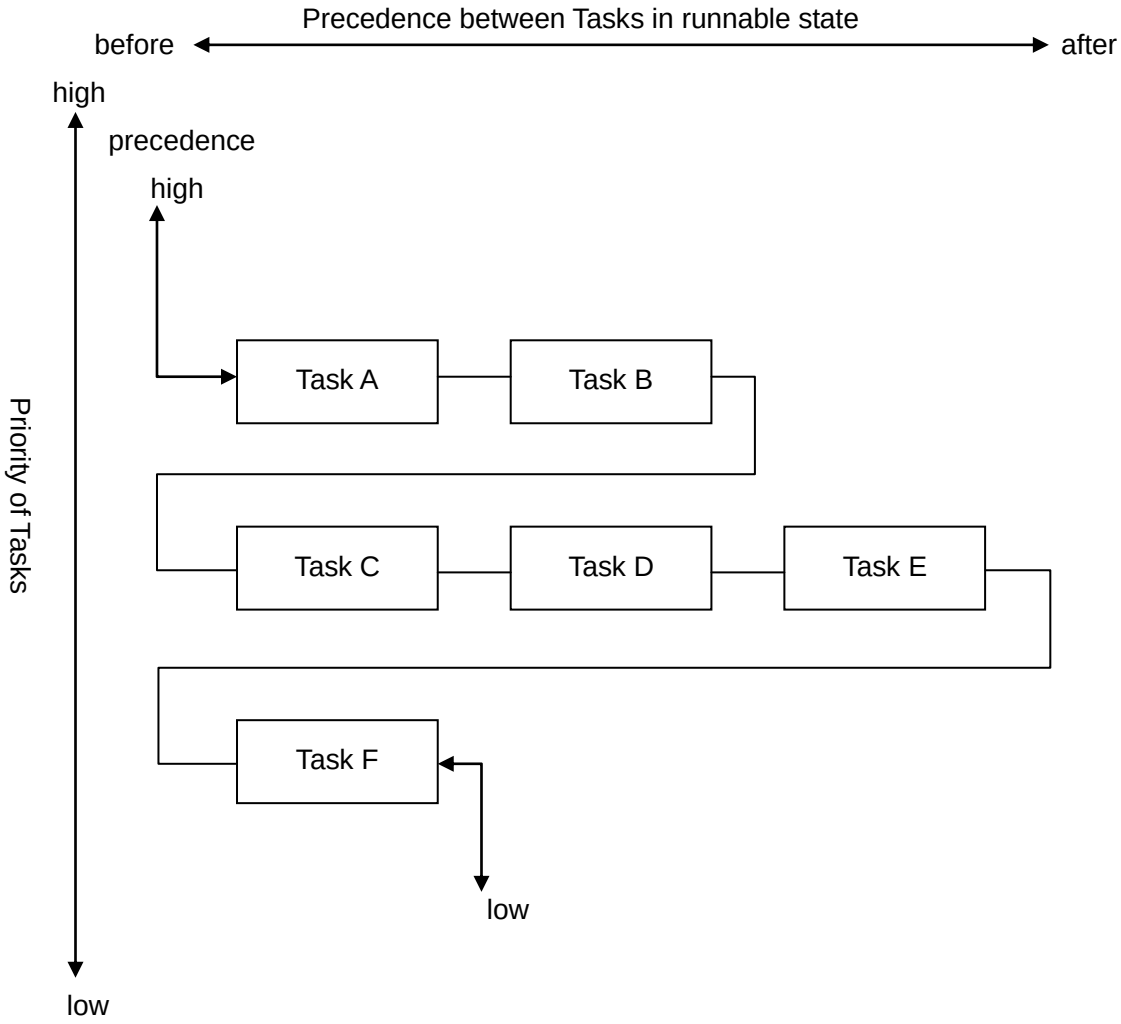
Releasing a task from waiting means that if the task is in WAITING state, it will be moved to the READY state, and if the task is in WAITING-SUSPENDED state, it will be moved to SUSPENDED state.

2. 2. 2 Scheduling rules

The preemptive priority-based scheduling is conducted based on the priorities assigned to tasks. If there are a number of tasks in runnable state, the highest precedence task will be in the RUNNING state. However, when dispatch does not happen, dispatch will be pended until switch to the state that dispatch happened.

The highest precedence task means that among tasks of the same priority, the task that becomes runnable state first has the highest precedence. This relation ship is shown in the following figure. However, the precedence between task of the same priority may change due to the invocation of system calls (chg_pri,rot_rdq).

For example, task A has them same priority with task B, but task A entered the runnable state earlier, the precedence of task A can be considered as higher than task B.



The relation ship of priority and precedence

Chapter 3 Function overview of μC3/Standard

3.1 Context and System state

3.1.1 Processing units and Contexts

The kernel of μC3/Standard is executed by the following processing units. In this specification document, the descriptions about interrupt handlers apply to interrupt service routines as well, unless a specific description about interrupt service routines is provided.

A. Interrupt handlers
    A.1 Interrupt service routines
B. Time event handlers
C. CPU exception handlers
D. Task
E. Idle

3.1.2 Task contexts and non-task contexts

Contexts that can be regarded as a part of processing of task are called task contexts, while other contexts are called non-task contexts. Non-task contexts are classified as interrupt handlers, time event handlers, and idle context.

In μC3/Standard, the system call is invoked from task contexts, the system is invoked from interrupt handlers, the system call is invoked from time event handlers and the system call is invoked from initialization handlers are distinguished from each other. It is impossible to invoke system call from Idle. Using the parameter which designates the invoking task and the system call which designates the invoking task are also prohibited from non-task contexts.

Besides, the system call that can move the invoking task to the blocked state also cannot be invoked.

3.1.3 CPU locked state

The system state is the state in either the CPU locked or CPU unlocked. In the CPU locked state, all interrupts are disabled except for non-kernel interrupts, and dispatch does not happen. Moreover, because interrupts are disabled, the activation of time event handlers is pended.

The transition to the CPU locked state is called “locking the CPU”, while the transition to the CPU unlocked state is called “unlocking the CPU”. In detail, the process of locking the CPU and unlocking the CPU or the state right after interrupt handler starts will be different depending on the processor, please refer to “The manual of processor-dependent parts”for more Description.

In the CPU locked state, if system calls that can move to the blocked state are invoked, an

E_CTX error will be returned.

The system is in the CPU unlocked state right after time event handlers start, if the system was changed to CPU locked state by the application, the system must be in the CPU unlocked state before returning from handlers.

The system is in the CPU locked state after tasks start. The application must return the system to the CPU unlocked state before the invoking task exits. However, the interrupt might be available even in CPU locked state. That relation is different depending on the processor, please refer to “The manual of processor-dependent parts”for more details.

3.1.4 Dispatching disabled state

The system state is the state in either dispatching disabled or dispatching enabled. In the dispatching disabled state, dispatching does not happen.

The transition to the dispatching disabled state is called “disabling dispatching”, while the transition to the dispatching enabled state is called “enabling dispatching”.

In the dispatching disabled state, if system calls that can move the invoking task which is invoked from task context to the blocked state are invoked, an E_CTX error will be returned. On the other hand, system calls that can be invoked from non-task contexts do not have restrictions even in the dispatching disabled state.

The start of interrupt handlers, time event handlers do not change the dispatching disabled/enabled state. After these handlers start, the dispatching disabled/enabled state of task context executing before will be saved. Besides, if system calls that can change dispatching disabled/enabled state in these handlers are invoked, an E_CTX error will be returned.

3.1.5 Idle state

If there are no tasks in runnable state, time event handlers or interrupt processing, idle will be started, this state is called “Idle state”. The system will be in CPU unlocked state, dispatching enabled state after idle starts. In µC3/Standard, idle state has its own independent contexts which are different from other contexts in characteristic. That different characteristic is not to save the contexts when changing into other contexts.

Because context is not saved, interrupt will happen while idle is running, and after non-task contexts start, the system does not return to the place which interrupt happened during the idle period. In case of changing to idle context, the system must be executed from the beginning of Idle.

Idle prepared by the kernel will be a simple loop without processing. In this idle, the user can define the idle functions by configuration.

3.1.6 Task state during Dispatch pending state

Dispatch does not happen during execution of processing with higher precedence than that of the dispatcher, and while in CPU locked state, interrupt level higher than task level, or in the dispatching disabled state. This state is called Dispatch pending state.

In the dispatch pending state, even the task with higher precedence than currently executing task happened, that high precedence task will not be dispatched. The dispatch for high precedence task will be pending until the system is in a state where dispatch can happen. While dispatch is pending, the task that has been running remains in the RUNNING state, while the tasks have to run remains in the READY state after the system is in a state where dispatch happen.

Task state during dispatch pending state is described below.

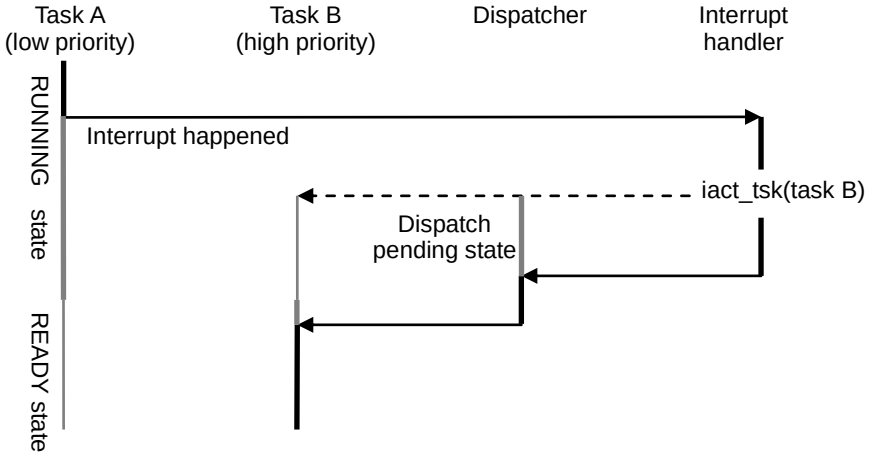


Figure of Dispatch pending state and Task state

Suppose that task B is activated from the interrupt service routine that was activated by the interrupt that happened during execution of task A, and the priority of task B is higher than the priority of task A. Since the precedence of the interrupt service routine is higher than that of dispatcher, the system is in the dispatch pending state while the interrupt service routine is executing after task B is activated, therefore dispatch does not happen. When the interrupt service routine execution terminates, the dispatcher is executed, and the task A is moved to READY state, task B is moved to RUNNING state.

Even after task B is activated in the interrupt handler, task A is in the RUNNING state and task B is in the READY state until the dispatcher is executed.

3.2 Delayed execution of System calls

In μC3/Standard, in order to reduce the interrupt handler overhead, and to minimize interrupt-inhibit section of the system call invoked by the non-interrupt, the delayed execution of system call that invoked by interrupt handler runs. In other words, in case of issuing the system call in the interrupt handler, only the error which depends on the object's state such as parameter check detected, and then the system call terminates at once. Then, the remaining system call will be handled after the interrupt has already terminated.

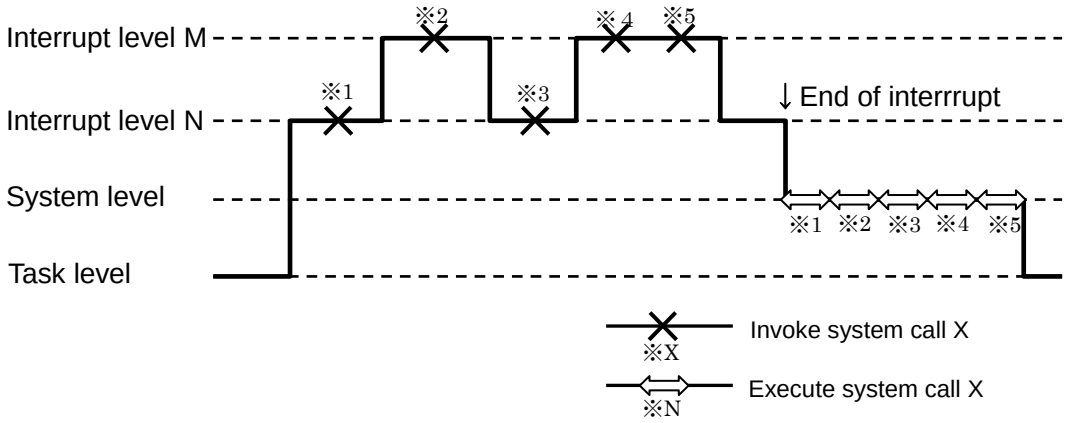


Figure of Delayed execution of System calls

3.2.1 System service block

In the delayed execution of system calls, the information of parameters will be memorized in FIFO order in the memory block that is called system service block (abbreviated as SSB) from invoking to executing. This system service block requires at least some system calls that invoked while an interrupt is processed, and there are 5 system service blocks are used in the figure as above. Besides, if the same interrupt is receipted for a multiple of times, more than system service blocks will be required until the system call runs.

3.2.2 Error detection of system call in delayed execution

In the delayed execution of system calls, because it is impossible to detect the error which is caused by the object's state at the point of time when the system call is being invoked, normal completion (E_OK) is returned. In order to detect the error has occurred during execution (except E_OK), the error handler is defined.

The delayed execution does not apply to the invoking of system call from time event handler classified as non-task context.

3.3 System start

System start is the function that can initialize μC3/Standard and starts multi-task states based on the configuration information.

- System start function (start_uC3)

μC3/Standard is activated based on the maximum ID number for each object, the area used for dynamic memory management and the other configuration information determined by the system design.

Initialization handler is described as shown below.

```
void initpr(void)
{
        Description of process required before transiting to multi-task
}
```

3.3.1 Configuration information of the maximum number of object ID

The maximum number of the object ID is shown as below.

- Maximum value of task ID
- Maximum value of semaphore ID
- Maximum value of event flag ID
- Maximum value of data queue ID
- Maximum value of mailbox ID
- Maximum value of mutex ID
- Maximum value of message buffer ID
- Maximum value of rendezvous port ID
- Maximum value of fixed-length memory pool ID
- Maximum value of variable-length memory pool ID
- Maximum value of cyclic handler ID
- Maximum value of alarm handler ID
- Maximum value of interrupt service routine ID
- Maximum value of standard COM driver ID

The maximum value of ID is the max ID number either specified or assigned at the generation time. In other words, this means the maximum number of objects which can be generated. 255 is the maximum number of tasks that can be generated, is also the maximum value of task ID, while the others is up to 999 and this is also the maximum value of ID for them.

Moreover, the maximum value of the standard COM driver ID is designated in case of using

μC3 standard COM driver.

3.3.2 Configuration information of memory management

The target areas of memory management are as below:

- System memory area
- Memory area for stack
- Memory area for memory pool

The memory area for memory pool is allocated for memory block area of fixed-length/variable length memory pool, data queue area and message buffer area. If NULL is specified, the memory area for memory pool will not be generated, the memory for stack will be used instead.

If not specified the stack area when the task is generated, the memory area for stack will be allocated from this area. If specified NULL, the memory area for stack will not be generated, in stead, the stack uses the system memory area.

The system memory area is allocated to the management areas that required when the objects are generated. This area must be specified, can not be omitted.

3.3.3 The other configuration information

In addition, there are items as below:

- The tick time that specifies the time tick cycle
- Maximum value of task priority
- Number of SSB
- User- defined Idle function
- Debug support

Time tick determines the time monitoring of the system call, and the time accuracy of time event handlers except for overrun handler. The lower value indicates a higher accuracy, but on the other hand overhead will also be bigger. It is specified in millisecond unit, usually between 1milisecond and 10 millisecond.

The highest task priority level is limited up to 31, thus, the maximum value will be specified less than 31.

User-defined idle function is described by the format as below and can be replaced by the standard idle function which written in μC3.

```
void user_idle(void)
{
        A necessary process in idle state
}
```

In debug support, there is initialization of stack area at the time of creating task, besides, there is no initialization, 0 initialization or initialization in task ID. Furthermore, the users can describe their own initialization function by the following format.

```
void user_stack_init(INT *stk, SIZE sz, ID tskid)
{
        Initialize byte size sz from the start address stk of task stack of ID number tskid
}
```

Other debug supports support the trace function when the contexts are switched or the system calls are issued. For more details, please refer to the manual of the debugger that supports for μC3/Standard.

3.4 Task management functions

Task management functions provide direct operation or reference to the task states. More specifically, task management functions include these below functions.

- Create a task (cre_tsk, acre_tsk)
- Delete a task (del_tsk)
- Activate a task (act_tsk, iact_tsk, sta_tsk)
- Terminate a task (ext_tsk, ter_tsk)
- Cancel activation request to task (can_act)
- Change task priority (chg_pri)
- Reference the state of a task (get_pri, ref_tsk, ref_tst)

The order of task execution is controlled by the base priority and the current priority. In case of using mutexes, there is a situation that the current priority is higher than the base priority, while if mutexes are not used, the current priority and the base priority are always equal. Simply, the task priority is mentioned refers to the current priority.

Activation requests for a task are queued. In other words, in case that want to reactivate a task has already activated, the activation request for this task is recorded. After that, when the task terminates, it will be automatically activated again. However, in the system call that activates a task with the specified start code (sta_tsk), the activation requests will not be queued. A task includes an activation request queuing count in order to realize the activation request queuing for a task. This activation request queuing count of task is clear 0 when the task is created.

When a task is activated, its extended information (exinf) is passed as a parameter. However, when a task is activated by the system call that activates a task with a start code (sta_tsk), the specified start code is passed instead of the extended information.

When a task is activated, the task's current priority and the task's base priority are initialized to the priority at the time of activation, the wakeup request count is cleared.

Task is described by the format as below.

```
void task(VP_INT exinf)
{
        Body of the task
        ext_tsk();
}
```

The behavior of a task returning from its main routine is identical to invoking ext_tsk.

Regarding the task management functions, the following macro is prepared.

TMAX_ACTCNT maximum activation request queuing count of task (999)

3.5 Task dependent synchronization functions

Task dependent synchronization functions provide the direct operation of task states to synchronize tasks. More specifically, these below functions are included.

- Put a task to the sleeping state (slp_tsk, tslp_tsk)
- Wake up a task from the sleeping state(wup_tsk, iwup_tsk)
- Cancel wakeup request (can_wup)
- Forcibly release a task from waiting state (rel_wai, irel_wai)
- Suspend a task (sus_tsk)
- Resume a task from the SUSPENDED state (rsm_tsk)
- Forcibly resume a task from the SUSPENDED state (frsm_tsk)
- Delay the execution of the invoking task (dly_tsk)

Wake up requests for a task are queued. In other words, in case that want to wake up a task is not in the sleeping state, the wake up request for this task is recorded, after that, when the task enters the sleeping state, the task will not be put in the sleeping state. A task includes a wake up request queuing count in order to realize the wake up request queuing for the task. The wake up request queuing of task is cleared to 0 when the task is activated.

Suspension requests for a task are nested. In other words, in case that want to put a task has already been in the SUSPENDED state (include WAITING-SUSPENDED state) in the SUSPENDED state again, the request for moving to SUSPENDED state of this task is recorded, after that, when the task is resumed from the SUSPENDED state (include WAITING-SUSPENDED state), suspension is not released. A task includes a suspension request nesting count in order to realize the suspension request nesting for the task. The suspension request nesting count of task is cleared to 0 when the task is activated.

Regarding to the task dependent synchronization functions, these below macros are prepared.

TMAX_WUPCNT	Maximum wakeup request queuing count of task (999)
TMAX_SUSCNT	Maximum suspension request nesting count of task(999)

3.6 Task exception handling

Task exception handling is not supported in this version.

3.7 Synchronization and Communication functions

Synchronization and communication functions provide synchronization and communication between tasks through objects that are independent of the tasks. The functions of semaphores, event flags, data queues, and mailboxes are included.

3.7.1 Semaphores

A semaphore is an object used for the exclusive control and synchronization when using resources through the representation of the availability and number of unused resources by a numerical value. More specifically, the following functions are included.

- Create a semaphore (cre_sem, acre_sem)
- Delete a semaphore (del_sem)
- Release semaphore resource (sig_sem, isig_sem)
- Acquire semaphore resource (wai_sem, pol_sem, twai_sem)
- Reference the state of a semaphore (ref_sem)

Semaphore has resource count expressing the availability and number of its resources, and queue in order of task priority or FIFO order of task waiting for resource acquisition. When a task releases resource, the resource count of semaphore is incremented by 1. In the other hand, when a task acquires resource, the resource count of semaphore is decremented by 1. In case that the resource count of semaphore was not enough (the case the resource count will be negative when decrementing), a task attempting to acquire a resource is placed in the queue, will be in the state of acquisition waiting for semaphore resource until a next resource is returned or till the permitted time. Moreover, in order to avoid the case where too many resources are returned to a semaphore, it is possible to set a maximum resource count for each semaphore. If more resources are returned to the semaphore than its maximum resource count (the case that if incrementing the resource count of a semaphore, the count will exceed the maximum resource count), an error will be returned.

Regarding to the semaphore functions, the following macro is prepared.

TMAX_MAXSEM Maximum value of the maximum resource count of semaphore (999)

3.7.2 Eventflags

Eventflag is an object used for synchronization through representing the availability of eventflags by the flag of each bit. More specifically, these below function are included.

- Create an eventflag (cre_flg,acre_flg)
- Delete an eventflag (del_flg)
- Set an eventflag (set_flg, iset_flg)

- Clear an eventflag (clr_flg)
- Wait for an eventflag (wai_flg, pol_flg, twai_flg)
- Reference the sate of an eventflag (ref_flg)

An eventflag has bit pattern expressing the availability of its events in each bit, and queue in order of task priority or FIFO order of task waiting for that eventflags. Sometimes the bit pattern of an eventflag is simply called an eventflag.

When a task notifies the event, it is able to set or clear specified bit pattern of eventflags. In the other hand, there is the waiting condition for a task waiting for an event. That is the task will wait till all bits specified (AND) or either of the specified bits (OR) in the eventflag bit pattern, in case that the specific condition has not been met, the task is placed in the queue, will be in the state of waiting for eventflags until the specific condition is met or till the permitted time.

Regarding to the eventflag functions, the below macro is prepared.

TBIT_FLGPTN The number of bit in an eventflag (depend on processor)

3.7.3 Data queues

A data queue is an object used for synchronization and communication by sending or receiving a one word message (this is called data). More specifically, these following functions are included.

- Create a data queue (cre_dtq,acre_dtq)
- Delete a data queue (del_dtq)
- Send a data to a data queue (snd_dtq,psnd_dtq,ipsnd_dtq,tsnd_dtq)
- Force-send a data to a data queue (fsnd_dtq,ifsnd_dtq)
- Receive a data from a data queue (rcv_dtq,prcv_dtq,trcv_dtq)
- Reference the state of a data queue (ref_dtq)

A data queue has an associated a sending queue in order of task priority or FIFO order of task waiting for sending data, and receiving queue in FIFO order of task waiting for receiving data. Moreover, a data queue has also ring- shaped data queue area used to store sent data.

A task sending a data places the data that wants to send in the data queue. In case that there is no space in the data queue area, the task is placed in the sending queue, will be in the state of sending waiting for data queue, until there is space for data queue area or till the permitted time.

In the other hand, a task receiving a data removes a data from a data queue. If there is no data in the data queue, the task is placed in the receiving queue, will be in the state of receiving waiting from a data queue,until a data is sent to a data queue or till the permitted time.

It is possible to realize the synchronous message function by setting the number of data that can be stored in the data queue area to 0. In other words, the sending side and the receiving

side waiting until the other invokes the system call of transceiver, and the data is transferred at the time which the system call was invoked.

Data is sent or received by a data queue can be an interger or the address of memory (pointer type). A data is sent or received is copied from the sender to the receiver.

In case that snd_dtq is invoked first, the task is moved to the state of sending waiting until rcv_dtq is invoked. Conversely, if rcv_dtq is invoked first, the task is moved to the state of receiving waiting until snd_dtq is invoked.

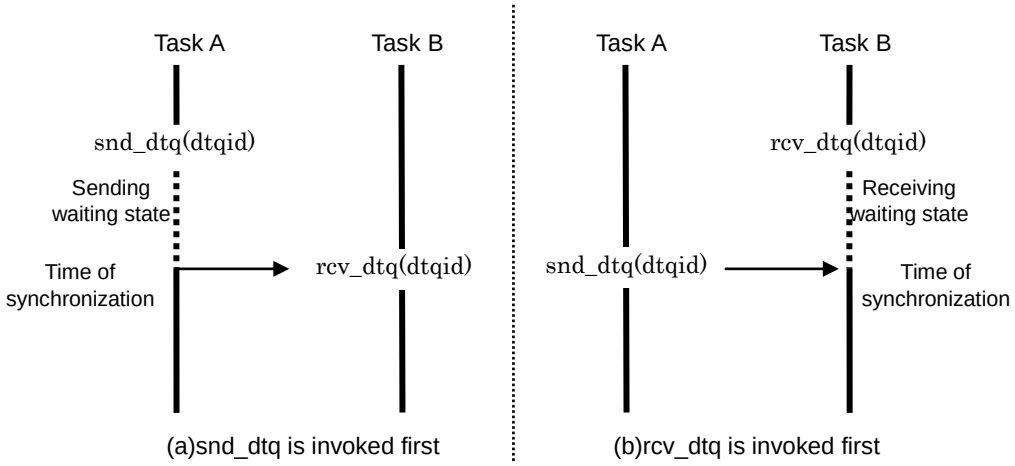


Figure of Synchronous communication through a data queue

Regarding to the data queue functions, the following macro is prepared.

SIZE dtqsz = TSZ_DTQ(UINT dtqcnt)

dtqcnt size of the data queue area necessary to store data (bytes)

3.7.4 Mailboxes

A mailbox is an object used for synchronization and communication by sending or receiving a message placed in the memory. More specifically, the following functions are included.

- Create a mailbox (cre_mbx,acre_mbx)
- Delete a mailbox (del_mbx)
- Send a message to a mailbox (snd_mbx)
- Receive a message from a mailbox (rcv_mbx,pol_mbx,trcv_mbx)
- Reference the state of a mailbox (ref_mbx)

A mailbox has an associated message queue and queue in order of task priority or FIFO order of task waiting to receiving a message. In case that sending message to the mailbox, if there is task in the state of receiving waiting, a message will be passed, then the task in waiting state is moved to the READY state. If there is no task in receiving waiting state, a message will

be placed in the message queue.

In case of receiving a message from a mailbox, if there is message in the message queue, the message will be removed. If there is no message, the task will be placed in the receiving waiting queue, and moved to receiving waiting state until a message is sent or till the permitted time. With mailbox functions, only the start address of the message placed in memory is actually passed, the content of passed message is not copied.

In sending message of mailbox, the start address of the message packet prepared by the application is used as a parameter. Moreover, when receiving a message, the start address of the message packet is received as a return parameter. More specifically, a message packet is the top area consisting of a message header which in kernel can specify the order of message queue, followed by an area store the message body used by the application.

The data type of message headers are defined as below:

T_MSG	Message header for a mailbox
T_MSG_PRI	Message header with priority for a mailbox

For example, the definition of message packet T_MSGPKT type in FIFO order as follows:

```
typedef struct t_msgpkt{
        T_MSG       msg;    /* message header */
                            /* message body is used by the application */
} T_MSGPKT;
```

The definition of message packet T_MSGPRIPKT type in the order of message priority as follows:

```
typedef struct t_msgpripkt{
        T_MSG_PRI   pk_msg; /* message header with priority */
                            /* message body is used by the application */
} T_MSGPRIPKT;
```

A user program is not allowed to modify the content of the massage packet placed in message queue. Moreover, a message packet already in a message queue must not be resent to a mailbox. If any infringement occurred in one of cases above, the message queue is destroyed, and it will lead to an fatal error.

Regarding to the mailbox functions, the following macro is prepared:。

SIZE mprihdsz = TSZ_MPRIHD(PRI maxmpri)

Size of the message queue header area for each priority necessary for mailbox with the maximum message priority of the messages to be sent/received is maxmpri (bytes)

【Recommendation】

A message packet may use the area statically allocated and a memory block dynamically allocated from a fixed-length or variable length memory pool. A common practice recommended that the sending task allocates a memory block from a memory pool, and sends the block as a message packet, while the receiving task release the memory block to the memory pool directly after removing the content of the message.

3.8 Extended synchronization and Communication function

Extended synchronization and communication function provide advanced synchronization and communication between task through objects that ra independent of the tasks. The function of mutexes, message buffers, and rendezvous are included.

3.8.1 Mutexes

A mutex is an object used for exclusive control among tasks when using shared resource. More specifically, these following functions are included:

- Create a mutex (cre_mtx, acre_ mtx)
- Delete a muxtex (del_ mtx)
- Lock a mutex (loc_ mtx, ploc_mtx, tloc_mtx)
- Unlock a mutex (unl_mtx)
- Reference the state of a mutex (ref_ mtx)

A mutex has the locked state and unlocked state, and a queue in order of the task priority or FIFO order of task waiting to lock. A mutex supports priority inheritance protocol and priority ceiling protocol to avoid unbounded priority inversions due to exclusive control.

A task will lock a mutex before using resources. In case a mutex is already locked by another task, the task will be moved to the state of lock waiting for the mutex untill the mutex is released or till the permitted time. The task has already been in the lock waiting state for the mutex is placed in the queue of mutex. A task unlock the mutex after using the resource.

A mutex uses the priority ceiling protocol can set the base priority of task with highest base priority among the tasks that may lock the mutex as the maximum priority when a mutex is created. In case that the task has base priority higher than the ceiling priority tries to lock a mutex with the priority ceiling protocol, an E_ILUSE error is returned. Moreover, in case that chg_pri is invoked to set the base priority of a task that has locked or is waiting to lock a mutex with the priority ceiling protocol to higher value than the ceiling priority of that mutex, chg_pri will return an E_ILUSE error.

In the priority control rule of µC3/Standard, the current priority of a task should be changed to equal to the highest of these priorities as below:

- The base priority of the task
- The current priority of the task has highest current priority among the tasks waiting to lock the mutexes in case that those mutexes with the priority inheritance protocol are locked by the task.
- The ceiling priority of the mutex has the highest ceiling priority among the mutexes in case that those mutexes with the priority ceiling protocol are locked by the task.

Here, the current priority of a task waiting for a mutex with the priority inheritance protocol is changed by another mutex operations or is changed by having its base priority changed by

chg_pri, the task that has the mutex locked have to have its current priority changed. Furthermore, if that task is waiting for another mutex with priority ceiling protocol, it also inherits the priority of the task that locked the mutex and its current priority changed.

If the current priority of a task has been changed by the mutex operations, the following processing will be executed. When a task whose priority has been changed is in the runnable state, the precedence of the task is changed according to the priority after changing. If the task that has the priority equal to the priority after changing has already existed, the resulting precedence will be lower than that task's precedence. When a task whose priority has been changed is in some task priority-ordered wait queue, the position in the wait queue is changed according to the new priority. If a task terminates while it still has mutexes locked by the task, all that mutexes will be unlocked.

A mutex with the FIFO order or task priority order has a similar functionality as a semaphore used in the exclusive control among tasks whose maximum resource count is 1. However, the differences are that a mutex can only be invoked form the task context, only be unlocked by the task that locked it and that a mutex is automatically unlocked when the task terminates.

3.8.2 Message buffer

A message buffer is an object used for synchronization and communication by sending or receiving a variable sized message. More specifically, these below functions are included:

- Create a message buffer (cre_mbf,acre_ mbf)
- Delete a message buffer (del_ mbf)
- Send a message to a message buffer (snd_mbf,psnd_mbf,tsnd_mbf,)
- Receive a message from a message buffer (rcv_mbf,prcv_mbf,trcv_mbf)
- Reference the state of a message buffer (ref_ mbf)

A message buffer has a sending queue in order of task priority or FIFO order of task waiting for sending a message and a waiting queue in FIFO order of task waiting to receiving a message. Also, a message buffer has also ring- shaped message buffer area used to store the sent messages. When sending a message, the message that want to send is copied into the message buffer. If there is no space in the message buffer area, the task will be placed in the sending queue of the message buffer, and in the state of sending waiting for a message buffer until there is space in the message buffer area or till the permitted time. In the other hand, when receiving the message, a message will be removed from the message buffer. If there is no message in the message buffer, the task will be placed in the receiving queue of the message buffer and in the state of receiving waiting for a message buffer until the next message is sent or till the permitted time.

It is possible to realize the synchronous message function by setting the size of the message buffer area to 0. In other words, the sending side and the receiving side waiting until the other invokes the system call, and the data is transferred at the time which the system call has been invoked by both of tasks.

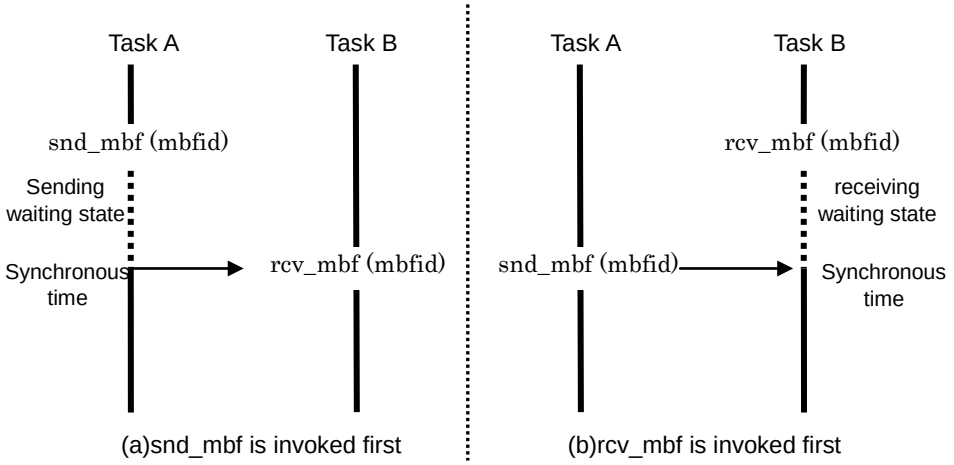


Figure of Synchronous communication through a message buffer

Regarding to the message buffer functions, the following macro is prepared.

SIZE mbfsz = TSZ_MBF(UINT msgcnt, UINT msgsz)

Size of the message buffer area necessary to store msgcnt messages each of size msgsz byte (approximate bytes)

3.8.3 Rendezvous

The rendezvous function is used for synchronous communication between tasks. It supports a procedure to handle a processing request from one task to another task and the return of the result to the requesting task. The object used for mutual waiting between two tasks is called rendezvous port. More specifically, the following functions are included.

- Create a rendezvous port (cre_por, acre_ por)
- Delete a rendezvous port (del_ por)
- Request a processing at a rendezvous port (cal_ por, tcal_ por)
- Accept a processing request at a rendezvous port (acp_ por, pacp_ por, tacp_ por)
- Return a processed result (fwd_ por)
- Forward a accepted processing request to another rendezvous port (rpl_rdv)
- Reference the state of a rendezvous port and a rendezvous (ref_por, ref_rdv)

Rendezvous port has a rendezvous invoking queue in order of task priority or FIFO order of task waiting for processing request and a rendezvous waiting queue in FIFO order of task waiting for processing request.

A task which requests a processing at a rendezvous port calls for a rendezvous by specifying a rendezvous port, a rendezvous condition, and a calling message that contains information about the requested processing. If the rendezvous has not been established、the task will be placed in the rendezvous calling queue and moved to the calling waiting state until the rendezvous is established or till the permitted time. If the rendezvous has been established, the task is released from the rendezvous calling queue and moved to the rendezvous termination waiting state.

In the other hand, the task that receives a processing request at a rendezvous port accepts the rendezvous by specifying the rendezvous port and the rendezvous condition. If the rendezvous has not been established, the task will be placed in the rendezvous accepting queue and moved to the accepting waiting state until the rendezvous is established or till the permitted time. In case that the rendezvous is established, the calling message is received, hen the task in the state of calling waiting is released from the queue, moved to the termination waiting state. At this time, the rendezvous number is assigned to distinguish the established rendezvous.

If a task that accepted the rendezvous completes the processing request, it returns the processed result to the calling task as a reply message, and the rendezvous is terminated. At the same time, the rendezvous calling task is moved to the READY state from the rendezvous termination waiting state.

A rendezvous condition is specified by a bit pattern. A rendezvous is established for some rendezvous port only when the bit patterns of the rendezvous conditions of both the calling task and the accepting task take the bit-wise AND and the result is not 0.

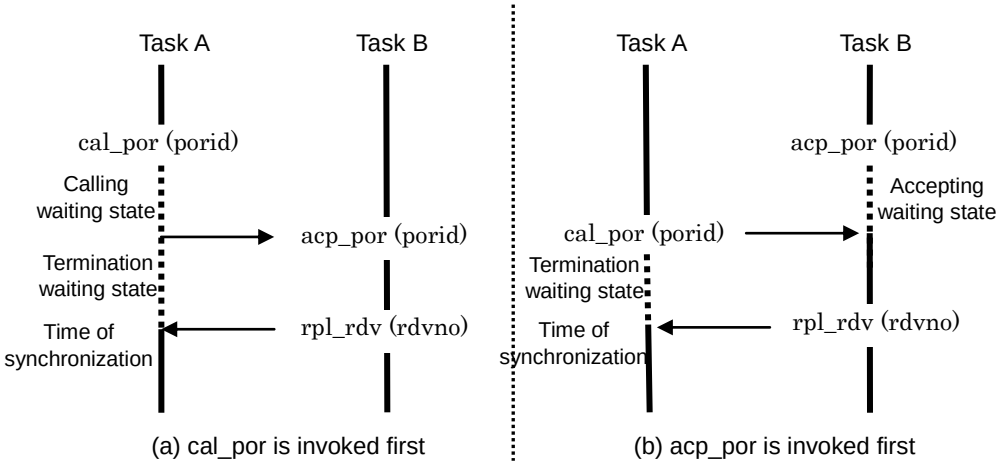


Figure of rendezvous operation

3.9 Memory pool management functions

Memory pool management functions provide dynamic memory management by software, include fixed-length memory pools and variable length memory pools.

3.9.1 Fixed-length memory pools

Fixed-length memory pool is an object used to manage fixed-size memory blocks dynamically. More specifically, the following functions are included.

- Create a fixed-length memory pool (cre_mpf, acre_mpf)
- Delete a fixed-length memory pool (del_mpf)
- Acquire a memory block from a fixed-length memory pool (get_mpf, pget_mpf, tget_mpf)
- Release a memory block to a fixed-length memory pool (rel_mpf)
- Reference the state of a fixed-length memory pool (ref_mpf)

A fixed-length memory pool has memory area where is used as fixed-length memory pool and a queue in order of the task priority or FIFO order of task that waiting to acquire a memory block. If there is no space in memory pool area, a task that acquires a memory block from the fixed-length memory pool will be placed in the queue and moved to the state of acquiring waiting for fixed-length memory block until the memory block is released or till the permitted time.

When the memory block is released, a user program will release the memory block to the fixed- length memory pool with the same ID number which acquired the memory block, and the start address of the acquired memory block have to be used as is when it is released. Also, the memory block that has already been released must not be released again. If any infringement occurred in one of cases above, the management information of the fixed-length memory pool is destroyed, and it will lead to an fatal error

Regarding to the fixed-length memory pool functions, the following macro is prepared.

SIZE mpfsz = TSZ_MPF(UINT blkcnt, UINT blksz)

Size of the fixed-length memory pool area necessary to be able to acquire blkcnt memory blocks each of size blksz bytes (in bytes)

3.9.2 Variable length memory pool

A variable length memory pool is an object used to manage the variable –sized memory block dynamically. More specifically, these below functions are included.

- Create a variable length memory pool (cre_mpf, acre_mpf)
- Delete a variable length memory pool (del_mpf)
- Acquire a memory block from a variable length memory pool (get_mpf, pget_mpf, tget_mpf)
- Release a memory block to a variable length memory pool (rel_mpf)
- Reference the state of a variable length memory pool (ref_mpf)

A variable length memory pool has memory area where is used as variable length memory pool and a queue in order of the task priority or FIFO order of task that waiting for acquiring a memory block. If there is no space in memory pool area, a task that acquires a memory block from the variable length memory pool will be placed in the queue and moved to the state of acquiring waiting for variable length memory block until the memory block is released or till the permitted time.

When the memory block is released, a user program will release the memory block to the variable length memory pool with the same ID number which acquired the memory block, and the start address of the acquired memory block have to be used as is when it is released. Also, the memory block that has already been released must not be released again. If any infringement occurred in one of cases above, the management information of the variable length memory pool is destroyed, and it will lead to an fatal error

> Regarding to the variable length memory pool functions,the following macro is prepared.
> SIZE mplsz = TSZ_MPL(UINT blkcnt, UINT blksz)
> Size of the variable length memory pool area necessary to be able to acquire blkcnt memory blocks each of size blksz bytes (approximate bytes)

In a variable length memory pool, the fragmentation of memory area of the variable length memory pool proceed by repeating the acquisition and release of memory blocks (i.e. being fragmented), thus, although sum total size of free space is sufficient, the contiguous memory is not available

3.10 Time management functions

Time management functions provide time-dependent processing. The time management functions include system time management, cyclic handlers, alarm handlers, overrun handler. Cyclic handlers, alarm handlers and overrun handler are generically called time event handlers.

3.10.1 System time management

System time management functions provide the operation of system time. More specifically, these below functions are included.

- Set or reference the system time (set_tim, get_tim)
- Supply time tick for updating system time (isig_tim)

System time is initialized to 0 when the system is started, and will be updated every time the system call (isig_tim) which supplies time tick is invoked by the application. Furthermore, system time is managed by millisecond unit, and all time specified by the system call is in millisecond unit excepting for the overrun handler function.

The processing of timeout, the releasing from the state of waiting for an elapsed time by dly_tsk, the activation of cyclic handler and the alarm handler depend on the system time

However, even system time is changed by setting system time (set_tim), time-out time of system call which has already been called, will not be changed.

3.10.2 Cyclic handler

A cyclic handler is a time event handler activated periodically. More specifically, the following functions are included.

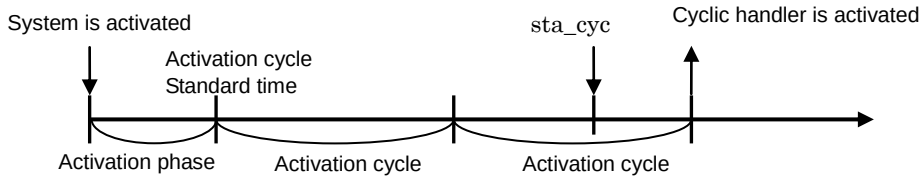
- Create a cyclic handler (cre_cyc,acre_cyc)
- Delete a cyclic handler (dle_cyc)
- Start a cyclic handle's operation (sta_cyc)
- Stop a cyclic handle's operation (stp_cyc)
- Reference the state of a cyclic handler (ref_cyc)

A cyclic handler is either in an operational state or a non-operational state.

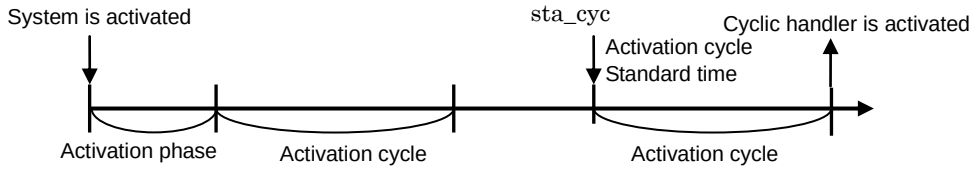
If the TA_STA attribute has been specified when a system is activated, a cyclic handler will be created in the operational state. However, in case that either TA_STA attribute or TA_PHS attribute has been specified, the time when the activation phase has been added to the system activation time is considered as the next activation time. In case neither of the attribute is specified, the time when the time invoking the system call which starts the cyclic handler's operation has been added to the activation cycle (sta_cyc) is considered as the next activation time. At the cyclic handler activation time, the cyclic handler is activated with its extended information (exinf) considered as a parameter. At this time, the next activation time is calculated

by adding the activation time of cyclic handler to the activation cycle.

When a cyclic handler is in a non-operational state with specified TA_PHS attribute, even the cyclic handler reaches to activation time, the cyclic handler is not activated, instead, only its next activation time is determined. When the system call that starts the cyclic handler ‘s operation (sta_cyc) is called, the cyclic handler is moved to an operational state and its next activation time is recalculated if necessary. When the system call that stops the cyclic handler’ operation (stp_cyc) is called, the cyclic handler is moved to a non-operation state.



(a) When the activation phase is preserved (TA_PHS spcedified)



(b) When the activation phase is not preserved (TA_PHS not specified)

Figure of Preserving activation phase

The activation cycle of the cyclic handler is the relative time which specifies the next activation time of the cyclic handler based on time which should have for activating cyclic handler (not the activated time). Therefore, each individual interval time when the cycle handler activated can possibly be shorter than the activation cycle, but in the long term, the average of activation interval will correspond with the activation cycle.

The n-th activation of the cyclic handler must occur after an elapsed time greater than (activation phase + activation cycle x (n – 1)) from the time when the system call that create the cyclic handler is invoked. For example, for a system with a 10ms time tick cycle, if the cyclic handler is created with the activation phase set to 15ms and the activation cycle set to 25ms,the system time of cyclic handler activation will be 20ms,40ms,70ms,90ms,and 120ms.

Cyclic handler is described by the format as below:

```
void cychdr(VP_INT exinf)
{
        Body of the cyclic handler
}
```

The restriction in μC3/Standard that can not specify the time which is shorter than time tick in the activation cycle.

3.10.3 Alarm handlers

An alarm handler is a time event handler activated at a specified time. More specifically, the following functions are included.

- Create an alarm handler (cre_alm, acre_alm)
- Delete an alarm handler (del_alm)
- Start an alarm handler's operation (sta_alm)
- Stop alarm handler's operation (stp_alm)
- Reference the state of an alarm handler (ref_alm)

An alarm handler is either in an operational state or a non-operational state. The time at which the alarm handler is activated, called the activation time of the alarm handler, can be set for each alarm handler. At the activation time, the alarm handler is activated with its extended information (exinf) considered as a parameter. After the alarm handler is created, the activation of the alarm handler is not set,and the alarm handler is moved to a non-operational state.

If the system call that starts the operation of an alarm handler (sta_alm) is invoked, the activation time of the alarm handler is set after the relative time specified, then the alarm handler is moved to an operational state. At this time, in case that the alarm handler has already been in a operational state, only the activation time of the alarm handler is reset.

When the alarm handler is activated, its activation time is released, the alarm handler is moved to a non-operational state. Therefore, the system call that starts the operation of the alarm handler (sta_alm) can be invoked in the alarm handler.

In case that the alarm handler is in the operational state, when the system call that stops the operation of the alarm handler (stp_alm) is invoked, its activation time is released, and the alarm handler is moved to a non-operational state.

Alarm handler is described by the format as below:

```
void almhdr(VP_INT exinf)
{
    Body of the alarm handler
}
```

3.10.4 Overrun handler

The overrun handler is a time event handler activated when a task has been executed by the processor longer than a specified time. More specifically, the following functions are included.

- Define the overrun handler (def_ovr)
- Start the operation of the overrun handler (sta_ovr)
- Stop the operation of the overrun handler (stp_ovr)
- Update the overrun handler's time (ivsig_ovr)
- Reference the state of the overrun handler (ref_ovr)

One overrun handler can be defined for the whole task, each task has its own timer that times the processing time used. This processing time used is the accumulated processing time consumed by the task from the operation of overrun handler starts.

When the consumed time exceeds the processing time limit set at the time of overrun handler activation, the overrun handler is activated with the task ID number and its extended information (exinf) considered as a parameter

Any unit of time can be used in the processing time used and the processing time limit, the system call that update the overrun handler‘s time (ivsig_ovr) is invoked at the interval of time accuracy requested according to the system design. In other words, when the system call that updates the overrun handler's time is invoked, the processor time using in case that overrun handler in the operational state is incremented by 1 (time unit) .

When the system call that starts the operation of the overrun handler (sta_ovr) is invoked, the specified time is set to the processing time limit, the processing time used is cleared to 0 and the overrun handler is moved to a operational state. If the system call that starts the overrun handler's operation is invoked while the overrun handler has already been in a operational state, it is reset.

Overrun handler is described by the following format:

```
void ovrhdr(IDtskid, VP_INTexinf)
{
    Body of the overrun handler
}
```

3.11 System state management functions

System state management function is a function used to update and reference the system state. More specifically, the following functions are included.

- Rotate task precedence (rot_rdq, irot_rdq)
- Reference the ID of the task is in the RUNNING state (get_tid, iget_tid)
- Lock and unlock the CPU (loc_cpu, iloc_cpu, unl_cpu, iunl_cpu)
- Enable and disable task dispatch (dis_dsp, ena_dsp)
- Reference the context and the system state (sns_ctx, sns_loc, sns_dsp, sns_dpn, ref_sys)

3.12 Interrupt management functions

Interrupt management functions provide management for interrupt service routines started by external interrupts. More specifically, the following functions are included.

- Define an interrupt handler (def_inh)
- Create or delete an interrupt service routine (cre_isr, acre_isr, del_isr)
- Reference the state of an interrupt service routine (ref_isr)
- Enable or disable an interrupt (dis_int, ena_int)
- Change or reference the interrupt mask (chg_ims, get_ims)

The following data types are used for interrupt management functions.

INHNO	interrupt handler number
INTNO	interrupt number
IMASK	interrupt mask

In μC3/Standard, the meaning of interrupt handler number and interrupt number are the same. The definition or the creation of an interrupt handler and interrupt service routine for the interrupt with the same interrupt number can not be done. Please find the Description of interrupt number allocation in “The manual of processor-dependent parts”.

The contents of interrupt mask data type IMASK are different depending on the processor’s architecture. Besides, the integration of the function to disable or enable an interrupt handler is different depending on the processor. For more details, please refer to “The manual of processor-dependent parts”.

Interrupt handler is described by the following format.

```
void inhhdr(void)
{
        Body of the interrupt handler
}
```

When an interrupt service routine is activated, the extended information (exinf) of the interrupt service routine is passed as a parameter. The format to write an interrupt service routine is shown below.

```
void isr(VP_INT exinf)
{
        Body of the interrupt service routine
}
```

3.13 System configuration management functions

System configuration management function is a function used for managing the system configuration and version information. More specifically, the following functions are included:

- Define a CPU exception handler (def_exc)
- Reference the configuration (ref_cfg)
- Reference the version information (ref_ver)

The following data type is use for system configuration management function.

EXCNO CPU exception handler number

The integration of CPU exception handler is different depending on the processor, there is case that is not integrated or has some limitation. For more details, please refer to “The manual of processor-dependent parts”.

CPU exception handler is described by the following format:

```
void exchdr(void)
{
        Body of the CPU exception handler
}
```

3.14 Peculiar functions

The unique functions in μC3/Standard include device driver management and error handler.

3.14.1 Device driver management

Device driver management is a function used to control and manage the standard COM driver. More specifically, the following functions are included.

- Define a device driver (vdef_dev)
- Control a device driver (vctr_dev)

The start address of device driver can be associated with the ID number by the device driver definition. In addition, in control, the device driver is invoked according to the function code and the control contents.

The format to write a device driver is shown below. The device driver is invoked with device control information packet ctrdev, function code funcid and control information packet ctrblk that can be obtained from the ID number of device driver considered as the parameter.

```
void devhdr(ID funid, VP ctrdev, VP ctrblk)
{
        Body of the device driver
}
```

3.14.2 Error handler

Error handler is a function used to detect the error of system calls during delayed execution. More specifically, the following function is included.

・Define an error handler (vdef_err)

If the error (excepting for E_OK) is occurred at the running time when the system call for delayed execution has been invoked from the interrupt handler, the error handler is invoked.

The format to write a error handler is shown below. The function code of the target's system call funcn, the occurred error code ercd, the array of arguments para when invoking the system call is invoked as parameters.

```
void err_hdl(FN funcn, ER ercd, VP_INT *para);
{
    Body of the error handler
}
```

The types of function codes are listed here below.

TFN_ACT_TSK	-0x07	iact_tsk (act_tsk)
TFN_STA_TSK	-0x09	sta_tsk
TFN_SET_FLG	-0x2B	iset_flg (set_flg)
TFN_SIG_SEM	-0x23	isig_sem (sig_sem)
TFN_PSND_DTQ	-0x36	ipsnd_dtq (psnd_dtq)
TFN_WUP_TSK	-0x13	iwup_tsk (wup_tsk)
TFN_REL_WAI	-0x15	irel_wai (rel_wai)

Chapter 4 Configuration and system activation

4.1 Kernel activation

To μC3 kernel, the system is initialized and multitask is started by the invoking for the activation function of kernel in the main function (or its similar functions).

start_uC3 Kernel activation

【Format】

ER ercd = start_uC3(T_CSYS *pk_csys, FP inihdr);

【Parameter】

T_CSYS *	pk_csys	Pointer to the packet containing the system creation information
FP	inihdr	The start address of the handler initialization

Content of pk_csys (T_CSYS type)
(Refer to 4.3 Kernel configuration)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (pk_csys is invalid)
E_NOMEM	Insufficient memory (object control area cannot be allocated)

【Description】

This system call creates a kernel based on the system creation information specified by pk_csys, and after the execution of initialization handler specified by inihdr, the system is moved to multi-task. This meant that, after a minimum required object is created by inihdr, and the initialization handler terminates, if there are tasks in READY state, the task with highest priority among tasks will take the control. In addition, inihdr is executed in the interrupt enabled state of non-task context.

4.2 System stack

In µC3/Standard, non-task contexts are executed on system stacks. As a result, the additional value of the stack size of time event handler which consumes the largest stack area, and of the stack size consumed by the interrupt processing (interrupt handler and interrupt service routine), is just an estimated value. Moreover, suppose the case with multiple interrupts, the stack size of interrupt handler which consumes the largest stack will be increased at each interrupt level.

4.3 Kernel configuration

μC3 kernel configures dynamically all objects based on the system creation information of the kernel's activation function.

T_CSYS type includes

tskpri_max	Maximum value of task priority (1 ~ 31)
tskid_max	Maximum value of task ID (1 ~ 255)
semid_max	Maximum value of semaphore ID (0~999)
flgid_max	Maximum value of eventflag ID (0~999)
dtqid_max	Maximum value of data queue ID (0~999)
mbxid_max	Maximum value of mailbox ID (0~999)
mtxid_max	Maximum value of mutex ID (0~999)
mbfid_max	Maximum value of message buffer ID (0~999)
porid_max	Maximum value of rendezvous port ID (0~999)
mpfid_max	Maximum value of fixed-length memory pool ID (0~999)
mplid_max	Maximum value of variable length memory pool task ID (0~999)
almid_max	Maximum value of alarm handler ID (0~999)
cycid_max	Maximum value of cyclic handler ID (0~999)
isrid_max	Maximum value of interrupt service routine ID (0~999)
devid_max	Maximum value of device driver ID (0~999)
tick	Time tick (unit of 1ms)
ssb_num	Number of system service blocks
sysmem_top	Top address of system memory area
sysmem_end	End address of system memory area (indicates last address +1)
stkmem_top	Top address of memory area for stack
stkmem_end	End address of memory area for stack (indicates last address +1)
mplmem_top	Top address of memory area for memory pool
mplmem_end	End address of memory area for memory pool (indicates last address +1)
ctrtim	Timer control function (for extended reserve)
sysidl	Idle processing
	SYSTEM_IDLE — Idle processing of the system
	USER_IDLE (func) — Idle function in user definition
inistk	Stack initialization processing
	STACK_NO_INIT — No initialization
	STACK_ZERO_INIT — Initializing by 0
	STACK_ID_INIT — Initializing by task ID

	STACK_USER_INIT(func)	Initialization function in user definition
trace	Trace	
	DISABLE_TRACE	No trace
agent	Agent	
	DISABLE_AGENT	No agent activation

Chapter 5 Description of System call

5.1 Task management function

cre_tsk	**Create task**
acre_tsk	**Create task (Automatic ID number assignment)**

【Format】

ER ercd = cre_tsk(ID tskid, T_CTSK *pk_ctsk) ;

ER_ID tskid = acre_tsk (T_CTSK *pk_ctsk) ;

【Parameter】

ID	tskid	ID number of the task to be created (except acre_tsk)
T_CTSK *	pk_ctsk	Pointer to the packet containing the task creation information

pk_ctsk includes (T_CTSK type)

ATR	tskatr	Task attribute
VP_INT	exinf	Task extended information
FP	task	Task start address
PRI	itskpri	Task initial priority
SIZE	stksz	Task stack area size (in bytes)
VP	stk	Task stack area start address
VB const *	name	Task name (character string)

【Return value】

In case of cre_tsk

ER	ercd	Successful completion (E_OK) or error code

In case of acre_tsk

ER_ID	tskid	ID number of the created task (positive value) or error code

【Error code】

E_PAR	Parameter error (itskpri is invalid)
E_ID	Invalid ID number (tskid is invalid or unusable ; only cre_tsk)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (stack area cannot be allocated)
E_NOID	Insufficient ID number (no allocatable task ID ; only acre_tsk)

E_OBJ	Object state error (target task is already registered ; only cre_tsk)

【Invoking context 】	cre_tsk	acre_tsk
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

These system calls creates a task with an ID number specified by tskid based on the task creation information specified by pk_ctsk. More specifically, the target task is moved from the NON-EXISTENT state to either the DORMANT state or the READY state. tskatr is the attribute of the task. exinf is the extended information passed as a parameter when the task is started. task is the start address of the task. itskpri is the task initial priority. stksz is the size of the stack area (in bytes). stk is the start address of the stack area.

acre_tsk assigns an ID number with biggest value from the pool of unassigned task IDs for the task to be created and returns the assigned ID number as a return value.

tskatr can be specified as (TA_HLNG | [TA_ACT]). If TA_ACT (= 0x02) is specified, the target task is moved to the READY state, the task's extended information is passed as a parameter when the task is started.

The memory area with stksz size from the address specified by stk is used as the stack area for the task execution. If stk is specified as NULL (= 0), the memory area with size specified by stksz is automatically allocated from the memory area used for defined stack by the configuration.

del_tsk Delete task

【Format】

ER ercd = del_tsk (ID tskid) ;

【Parameter】

ID	tskid	ID number of the target task to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from nom- task)
E_OBJ	Object state error (specified task is not in the DORMANT state)
E_NOEXS	Non-existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the task with an ID number specified by tskid. In other words, the specified task is moved from the DORMANT state to the NON-EXISTENT state.

If the specified task is not in the DORMANT state, an E_OBJ error is returned, while if the specified task is in the NON-EXISTENT state E_NOEXS error is returned.

act_tsk	**Active task**
iact_tsk	

【Format】

ER ercd = act_tsk (ID tskid) ;

ER ercd = iact_tsk (ID tskid) ;

【Parameter】

ID	tskid	ID number of the task to be activated

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid, or unusable)
E_NOMEM	Insufficient memory (SSB is lack：only invoked from interrupt handler)
E_NOEXS	Non-existent object (specified task is not registered)
E_QOVR	Queuing overflow (overflow of activation request queuing count)

【Invoking context】	act_tsk	iact_tsk
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

This system call activates the task with an ID number specified by tskid. In other words, the specified task is moved from the DORMANT state to the READY state. The extended information of the task is passed as a parameter when the task is activated.

If the specified task is not in the DORMANT state, the activation request for the task is queued. Specifically, the activation request queuing count is incremented by 1. If the count then exceeds the maximum possible count, an E_QOVR error is returned. However, if this system call is invoked from the interrupt handler and has its execution delayed, an E_QOVR error can not be detected in the error code of the return values. If TSK_SELF (= 0) is specified, the invoking task is specified. However, if TSK_SELF is specified when this system call is invoked from non-task contexts, an E_ID error is returned.

【Recommendation】

act_tsk and iact_tsk in μC3/Standard are the same system call, and both can be used by the same method regardless of the invoking context. However, it is recommended that act_tsk is used when invoking from the task context and iact_tsk should be used for the remains.

can_act Cancel task activation request

【Format】

ER_UINT actcnt = can_act (ID tskid) ;

【Parameter】

ID	tskid	ID number of task for cancelling activation request

【Return value】

ER_UINT	actcnt	Activation request count is queued (positive value or 0) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call clears all activation requests queuing count for the task with an ID number specified by tskid and returns the activation request queuing count before clearing for the task.

If tskid is TSK_SELF (= 0), the invoking task is specified. However, if TSK_SELF is specified when this system call is invoked from non-task contexts, an E_ID error is returned.

sta_tsk Activate task (with a start code)

【Format】

ER ercd = sta_tsk (ID tskid, VP_INT stacd) ;

【Parameter】

ID	tskid	ID number of the task to be activated
VP_INT	stacd	Task start code

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid, or unusable)
E_NOMEM	Insufficient memory (SSB is lack : only invoked from interrupt handler)
E_OBJ	Object state error (specified task is not in DORMANT state)
E_NOEXS	Non-existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Possible

【Description】

This system call activates the task with an ID number specified by the tskid. In other words, the specified task is moved from the DORMANT state to the READY state. The start code (stacd) is passed as a parameter when the task is activated.

If the specified task is not in the DORMANT state, the system call does not queue a activation request for the task, an E_OBJ error is returned. However, if this system call is invoked from the interrupt handler and has its execution delayed, an E_OBJ error can not be detected in the error code of the return values

【Recommendation 】

Sta_tsk exists for the compatibility with μITRON3.0 specification. In μC3/Standard, it is recommended that act_tsk and iact_tsk should be used instead of sta_tsk.

ext_tsk Terminate invoking task

【Format】

void ext_tsk ( ) ;

【Parameter】

None

【Return value】

None

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call terminates the invoking task. In other words, the invoking task is moved from the RUNNING state to the DORMANT state. If the activation request queuing count for the invoking task is 1 or more, the count is decremented by 1 and the invoking task is moved to the READY state. At this time, task priority initialization, wakeup request counter count cancellation, stack pointer initialization are executed as the processing that must be taken at task activation time. The extended information of the task is passed as a parameter when the task is activated.

If the system call is invoked from a task context, this system call never returns. However, in case of non-task context, this system call will return with no error code is returned.

exd_tsk Terminate and delete invoking task

【Format】

```
void exd_tsk ( ) ;
```

【Parameter】

None

【Return value】

This system call does not return

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call terminates and deletes the invoking task. In other words, the invoking task is moved from the RUNNING state to the NON-EXISTENT state, and the actions that must be taken at task termination and task deletion time are performed.

If the system call is invoked from a task context, this system call never returns. However, in case of non-task context, this system call will return with no error code is returned.

ter_tsk Forcibly terminate task

【Format】

ER ercd = ter_tsk (ID tskid) ;

【Parameter】

ID	tskid	ID number of task to be forcibly terminated

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_ILUSE	Illegal system call use (specified task is an invoking task)
E_OBJ	Object state error (specified task is in the DORMANT state)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call moves the task will ID number specified by tskid to the DORMANT state forcibly. However, If the activation request queuing count for the specified task is 1 or more, the count is decremented by 1 and the task is moved to the READY state. At this time, task priority initialization, wakeup request counter count cancellation, stack pointer initialization are executed as the processing that must be taken at task activation time. The extended information of the task is passed as a parameter when the task is activated.

If the specified task is in the DORMANT state, an E_OBJ error is returned. In addition, this system call can not terminate the invoking. If the specified task is a invoking task, an E_ILUSE error is returned.

chg_pri Change task priority

【Format】

ER ercd = chg_pri (ID tskid, PRI tskpri) ;

【Parameter】

ID	tskid	ID number of task whose priority is to be changed
PRI	tskpri	Base priority after changing

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (tskpri is invalid)
E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_ILUSE	Illegal system call use (refer to Description)
E_OBJ	Object state error (specified task is in the DORMANT state)
E_NOEXS	Non- existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call changes the base priority of the task with an ID number specified by tskid to the priority value specified by tskpri. If tskid is TSK_SELF (=0), the invoking task will be the specified task. In addition, if tskpri is TPRI_INI (=0), the base priority of the specified task is changed to the task's initial priority. In case that the current priority of the invoking task does not match the priority control rule of μC3/Standard caused by the mutex is locked, the current priority of tasks (including non-invoking tasks) is changed appropriately.

If the current priority has been changed when the specified task is in the runnable state, the task's precedence is changed to reflect the new priority. The specified task will have the lowest precedence among tasks with the same priority with the new priority.

If the task locked or is waiting to lock the mutexes with the priority ceiling protocol, and the priority is higher than these ceiling priorities, an E_ILUSE error is returned.

get_pri Reference task priority

【Format】

ER ercd = get_pri (ID tskid, PRI *p_tskpri) ;

【Parameter】

ID	tskid	ID number of the task to reference

【Return value】

ER	ercd	Successful completion (E_OK) or error code
PRI	tskpri	Current priority of specified task

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_OBJ	Object state error (specified task is in the DORMANT state)
E_NOEXS	Non- existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the current priority of the task with an ID number specified by tskid and then returns it through tskpri.

If tskid is TSK_SELF (=0), the invoking task will be the target Task.

【Usage】

```
PRI tskpri;          /* the area where store the task's current priority is allocated */
ER ercd;
                     /* invoke the pointer to the storing area as a parameter */
ercd = get_pri(ID_Task1, &tskpri);
```

ref_tsk Reference task state

【Format】

ER ercd = ref_tsk(ID tskid, T_RTSK *pk_rtsk) ;

【Parameter】

ID	tskid	ID number of the task to be referenced
T_RTSK *	pk_rtsk	Pointer to the packet returning the task state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rtsk includes (T_RTSK type)

STAT	tskstat	Task state
PRI	tskpri	Task current priority
PRI	tskbpri	Task base priority
STAT	tskwait	Reason for waiting
ID	wobjid	Object ID number for which task is waiting
TMO	lefttmo	Remaining time until timeout
UINT	actcnt	Activation request queuing count
UINT	wupcnt	Wakeup request queuing count
UINT	suscnt	Suspension request nesting count

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non- existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the task with an ID number specified by tskid, then the state is returned through the packet specified by pk_rtsk.

One of the following values is returned through tskstat due to the state of the specified task.

TTS_RUN	0x01	RUNNING state
TTS_RDY	0x02	READY state
TTS_WAI	0x04	WAITING state

TTS_SUS	0x08	SUSPENDED state
TTS_WAS	0x0C	WAITING-SUSPENDED state
TTS_DMT	0x10	DORMANT state

If the specified task is not in the DORMANT state, the current priority of the task is returned through tskpri, and the base priority is returned through tskbpri. If the specified task is in the DORMANT state, the values returned through them are undefined.

If the specified task is in the WAITING state, one of the following values is returned through tskwait due to the reason of the task's waiting state. If the task is not in the WAITING state, the undefined value is return.

TTW_SLP	0x0001	Sleeping state
TTW_DLY	0c0002	Waiting state for an elapsed time
TTW_SEM	0x0004	Waiting state for acquiring a semaphore resource
TTW_FLG	0x0008	Waiting state for an eventflag
TTW_SDTQ	0x0010	Waiting state for sending to a data queue
TTW_RDTQ	0x0020	Waiting state for receiving from a data queue
TTW_MBX	0x0040	Waiting state for receiving from a mailbox
TTW_MTX	0x0080	Waiting state to locking a mutex
TTW_SMBF	0x0100	Waiting state for sending to a message buffer
TTW_RMBF	0x0200	Waiting state for receiving from a message buffer
TTW_CAL	0x0400	Waiting state for calling a rendezvous
TTW_ACP	0x0800	Waiting state for accepting a rendezvous
TTW_RDV	0x1000	Waiting state for terminating a rendezvous
TTW_MPF	0x2000	Waiting state for acquiring a fixed-length memory block
TTW_MPL	0x4000	Waiting state for acquiring a variable length memory block

If the task is in the WAITING state, the object ID number of the task is waiting for is returned through wobjid which is neither in the sleeping state nor the waiting state for an elapsed time. The undefined value is returned through wobjid in the remains.

When the specified task is in the WAITING state but not in the waiting state for an elapsed time, the amount of time remaining for the task to timeout is returned through lefttmo. Specifically, the value calculated by subtracting the current time from the time at which the task will timeout is returned. However, the return value of lefttmo must be the amount of time until timeout and less than the actual amount of time until timeout. Therefore, if the timeout happens at the next time tick, 0 is returned through lefttmo. If the task is in the WAITING state forever (waiting without a timeout),TMO_FEVR is returned through lefttmo. If the task is not in the WAITING state or is in the waiting state for an elapsed time, the value returned through lefttmo is undefined.

The activation request queuing count for the specified task is returned through actcnt, the wake up request queuing count for the task is returned through wupcnt and the suspension request nesting count is returned through suscnt

If tskid is TSK_SELF (＝0) , the invoking task will be the specified task.

【Usage】

```
T_RTSK rtsk;            /* the area where store the task state is allocated */
ER ercd;
                        /* invoke the pointer to the storing area as a parameter */
ercd = ref_tsk (ID_Task1, & rtsk);
```

ref_tst Reference task state (Simplified version)

【Format】

ER ercd = ref_tst(ID tskid, T_RTST *pk_rtst) ;

【Parameter】

ID	tskid	ID number of the task to be referenced
T_RTST *	pk_rtst	Pointer to the packet returning the task state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rtst includes (T_RTST type)

STAT	tskstat	Task state
STAT	tskwait	Reason for waiting

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non- existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the minimum state for the task with an ID number specified by tskid, then the state is returned through the packet specified by pk_rtsk. This system call is a simplified version of ref_tsk, the same values returned by ref_tsk are returned through tskstat and tskwait.

If tskid is TSK_SELF (＝0), the invoking task will be the specified task.

5.2 Task dependent synchronization functions

slp_tsk	**Waiting for wakeup**
tslp_tsk	**Waiting for wakeup (with timeout)**

【Format】

ER ercd = slp_tsk ( ) ;

ER ercd = tslp_tsk (TMO tmout) ;

【Parameter】

TMO	tmout	Specified timeout (only for tslp_tsk)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_CTX	Context error (the state of preserving dispatch, except task)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting)
E_TMOUT	Polling failure or timeout (tslp_tsk)

【Invoking context】	slp_tsk	tslp_tsk
Task	Possible	Possible
Initialization handler	Impossible	Impossible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

This system call moves the invoking task to the sleeping state. However, if the wakeup request queuing count for the invoking task is 1 or more, the count is decremented by 1, the invoking task continues execution without entering the sleeping state.

tslp_tsk is the system call as slp_tsk with an additional timeout feature. Moreover, tmout can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, tslp_tsk that set tmout to TMO_FEVR will be used as slp_tsk

wup_tsk	Wake up task
iwup_tsk	

【Format】

ER ercd = wup_tsk (ID tskid) ;

ER ercd = iwup_tsk (ID tskid) ;

【Parameter】

ID	tskid	ID number of the task to be woken up

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_NOMEM	Insufficient memory (SSB is lack : only invoked from interrupt handler)
E_OBJ	Object state error (specified task is in the DORMANT state)
E_NOEXS	Non-existent object (specified task is not registered)
E_QOVR	Queuing overflow (overflow of wakeup request queuing count)

【Invoking context】	wup_tsk	iwup_tsk
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

This system call wakes up the task with an ID number specified by tskid from the sleeping state. E_OK will be returned to the task which has been released from waiting as a return value of the system call in WAITING state. In case the specified task is neither in sleeping state nor DORMANT state, the wakeup request queuing count for the task is incremented by 1. However, If the count calculated by adding 1 then exceeds the maximum value of wake up request queuing count, an E_QOVR error is returned. Also, if the specified task is in the DORMANT state, an E_OBJ error is returned. However, if this system call is invoked from the interrupt handler and has its execution delayed, an E_QOVR error and an E_OBJ error can not be detected in the error code of return values. If tskid is TSK_SELF (= 0), the invoking task will be the specified task. Nevertheless, if TSK_SELF is specified when this system call is invoked from non-task contexts, an E_ID error is returned.

【Recommendation】

wup_tsk and iwup_tsk in µC3/Standard are the same system call, thus, both can be used by the same method irregardless of the invoking context. However, it is recommended that wup_tsk is used when invoking from the task context and iwup_tsk should be used for the remains.

can_wup Cancel task wakeup request

【Format】

ER_UINT wupcnt = can_wup (ID tskid) ;

【Parameter】

ID	tskid	ID number of the task for cancelling wake up request

【Return value】

ER_UINT	wupcnt	Wake up request count is queued (positive value or 0) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_OBJ	Object state error (specified task is in the DORMANT state)
E_NOEXS	Non-existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call clears all wake up request queuing count for the task with an ID number specified by tskid, and returns the wake up request queuing count before clearing as the return value.

If the specified task is in the DORMANT state, an E_OBJ error is returned. If tskid is specified as TSK_SELF (＝0), the invoking task will be the specified task.

rel_wai	Forced release from waiting
irel_wai	

【Format】

ER ercd = rel_wai (ID tskid) ;

ER ercd = irel_wai (ID tskid) ;

【Parameter】

ID	tskid	ID number of the task to be forcibly released from waiting

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_NOMEM	Insufficient memory (SSB is lack : only invoked from interrupt handler)
E_OBJ	Object state error (specified task is not in WAITING state)
E_NOEXS	Non-existent object (specified task is not registered)

【Invoking context】	rel_wai	irel_wai
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

This system call forcibly moves the task in the WAITING state with an ID number specified by tskid to the READY state. When the task is released from waiting by this system call, an E_RLWAI error will be returned as a return value of the system call that placed the task in the WAITING state. In case the specified task is not in the WAITING state, an E_OBJ error is returned. However, if this system call is invoked from the interrupt handler and has its execution delayed, an E_OBJ error can not be detected in the error code of return values.

【Recommendation】

rel_wai and irel_wai in µC3/Standard are the same system call, thus, both can be used by the same method irregardless of the invoking context. However, it is recommended that rel_wai is used when invoking from the task context and irel_wai should be used for the remains.

sus_tsk Suspend task

【Format】

ER ercd = sus_tsk (ID tskid) ;

【Parameter】

ID	tskid	ID number of the task to be suspended

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (reference to Description)
E_OBJ	Object state error (specified task is in DORMANT state)
E_NOEXS	Non-existent object (specified task is not registered)
E_QOVR	Queuing overflow (overflow of suspension request nesting count)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call suspends the task with an ID number specified by tskid and task execution will be abandoned. Specifically, if the specified task is in the runnable state,it is moved to SUSPENDED state, if the task is in the WAITING state, it is moved to WAITING-SUSPENDED state. At the same time, the suspension request nesting count for the task is incremented by 1. If the count calculated by adding 1 then exceeds the maximum value of suspension request nesting count, an E_QOVR error is returned.

This system call can be invoked in the dispatching disabled state, but in the dispatching disabled state, if the invoking task is invoked as the specified task, an E_CTX error is returned. Moreover, even if the system call is invoked from the interrupt handler, an E_CTX error is also returned.

If tskid is TSK_SELF(=0), the invoking task will be the specified task.

rsm_tsk	Resume suspended task
frsm_tsk	Forcibly resume suspended task

【Format】

ER ercd = rsm_tsk (ID tskid) ;

ER ercd = frsm_tsk (ID tskid) ;

【Parameter】

ID	tskid	ID number of the task to be resumed

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_OBJ	Object state error (specified task is not in the SUSPENDED state)
E_NOEXS	Non-existent object (specified task is not registered)

【Invoking context】	rsm_tsk	frsm_tsk
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Impossible	Impossible

【Description】

This system call releases the task with an ID number specified by tskid from the SUSPENDED state, thus, the task execution is resumed.

rsm_tsk decrements the suspension request nesting count of the specified task by 1,if the count after decrement becomes 0, the task is moved to the READY state if it is in SUSPENDED state, and if the task is in WAITING-SUSPENDED state, it is moved to the WAITING state. If the count after decrement remains to be 1 or more, the state of the task is not changed. frsm_tsk clears the suspension request nesting count for the task to 0,and the task is moved to the READY state if it is in the SUSPENDED state, and if the task is in the WAITING-SUSPENDED state, it is moved to the WAITING state.

If the task is in the NON-EXISTENT state, an E_NOEXS error is returned, and if the task is neither in SUSPENDED state nor WAITING-SUSPENDED state, an E_OBJ error is returned.

dly_tsk Delay invoking task

【Format】

ER ercd = dly_tsk (RELTIM dlytim) ;

【Parameter】

RELTIM	dlytim	Amount of time to delay the invoking task (relative time)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_CTX	Context error (except task or the state that dispatch is preserved)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call moves the invoking task to the delayed state for the amount of time specified by dlytim since it is invoked. If the invoking task is set to be released from waiting when the specified time has passed, this system call completes normally and returns E_OK.

5.3 Task exception handling

Task exception handling is not supported in this version.

5.4 Synchronization and communication functions

5.4.1 Semaphore

cre_sem	**Create semaphore**
acre_sem	**Create semaphore (Automatic ID assignment)**

【Format】

ER ercd = cre_sem(ID semid, T_CSEM*pk_csem) ;

ER_ID semid = acre_sem(T_CSEM*pk_csem) ;

【Parameter】

ID	semid	ID number of the semaphore to be created (except acre_sem)
T_CSEM*	pk_csem	Pointer to packet containing the semaphore creation information

pk_csem includes (T_CSEM type)

ATR	sematr	Semaphore attribute
UINT	isemcnt	Initial semaphore resource count
UINT	maxsem	Maximum semaphore resource count
VB const *	name	Semaphore name (character string)

【Return value】

In case of cre_sem

ER	ercd	Successful completion (E_OK) or error code

In case of acre_sem

ER_ID	semid	ID number of the created semaphore (positive value) or error code

【Error code】

E_RSATR	Reserved attribute (sematr is invalid or unusable)
E_PAR	Parameter error (isemcnt, maxsem is invalid)
E_ID	Invalid ID number (semid is invalid or unusable ; only cre_sem)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (control area can not be allocated)
E_NOID	Lack of ID number (there is no semaphore ID assignable ; only acre_sem)
E_OBJ	Object state error (specified semaphore is already registered ; only cre_sem)

【Invoking context】	cre_sem	acre_sem
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

This system call creates a semaphore with an ID number specified by semid based on the semaphore creation information specified by pk_csem. sematr is the attribute of the semaphore, isemcnt is the initial value of the resource count after creation of the semaphore, maxsem is the maximum resource count of the semaphore.

acre_sem assigns an ID number with biggest value from the pool of unassigned semaphore IDs for the semaphore to be created and returns the assigned ID number as a return value

sematr can be specified as (TA_TFIFO || TA_TPRI). If TA_TFIFO (＝0x00) is specified, the semaphore's wait queue will be in FIFO order, if TA_TPRI (＝0x01) is specified, the semaphore's wait queue will be in task priority order.

When a value greater than maxsem is specified in isemcnt, an E_PAR error is returned. Moreover, if maxsem is specified as 0 or a value greater than the maximum value for the maximum semaphore resource count (TMAX_MAXSEM), an E_PAR error is returned.

del_sem Delete semaphore

【Format】

ER ercd = del_sem(ID semid) ;

【Parameter】

ID	semid	ID number of the semaphore to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (semid is invalid or unusable ; only cre_sem)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified semaphore is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the semaphore with an ID number specified by semid.

sig_sem	Release semaphore resource
isig_sem	

【Format】

ER ercd = sig_sem(ID semid) ;

ER ercd = isig_sem(ID semid) ;

【Parameter】

ID	semid	ID number of the semaphore to which resource is released

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (semid is invalid or unusable ; only cre_sem)
E_NOMEM	Insufficient memory (lack of SSB: only for invoking from the interrupt handler)
E_NOEXS	Non-existent object (specified semaphore is not registered)
E_QOVR	Queuing overflow (release will exceed maximum resource count)

【Invoking context】	sig_sem	isig_sem
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

This system call releases the task at the head of the queue from waiting if there is a task waiting to acquire the resource for the semaphore with ID number specified by semid, and then moves to the READY state. At this time, the specified semaphore resource count is not changed. In addition, the released task receives an E_OK as the return value of the system call that placed the task into the WAITING state.

If no task are waiting to acquire the resource, the semaphore resource count is incremented by 1. At this time, if the count is calculated by adding 1 then exceeds the maximum semaphore resource count, an E_QOVR error is returned. However, if this system call is invoked from the interrupt handler and has its execution delayed, an E_QOVR error can not be detected in the error code of return values.

【Recommendation】

sig_sem and isig_sem in μC3/Standard are the same system call, and both can be used by the same method irregardless of the invoking context. However, it is recommended that sig_sem is used when invoking from the task context and isig_sem should be used for the remains.

wai_sem	**Acquire semaphore resource**
pol_sem	**Acquire semaphore resource (polling)**
twai_sem	**Acquire semaphore resource (with timeout)**

【Format】

```
ER ercd = wai_sem(ID semid) ;
ER ercd = pol_sem(ID semid) ;
ER ercd = twai_sem(ID semid, TMO tmout) ;
```

【Parameter】

ID	semid	ID number of the semaphore from which resource is acquired
TMO	tmout	Specified timeout (only twai_sem)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (tmout is invalid ; only twai_sem)
E_ID	Invalid ID number (semid is invalid or unusable)
E_CTX	Context error (invoked from others except task : except pol_sem, invoked from the interrupt handler: only pol_sem the state of preserving dispatch: except pol_sem)
E_NOEXS	Non-existent object (specified semaphore is not registered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting; except pol_sem)
E_TMOUT	Polling failure or timeout (except wai_sem)
E_DLT	Delete waiting object (semaphore is deleted while waiting; except pol_sem)

【Invoking context】	wai_sem	pol_sem	twai_sem
Task	Possible	Possible	Possible
Initialization handler	Impossible	Possible	Impossible
Time event handler	Impossible	Possible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

This system call acquires one resource from the semaphore with an ID number specified by semid. If the resource count of the specified semaphore is 1 or more, the semaphore resource

count is decremented by 1, then complete this system call without entering the WAITING state. If the resource count of the specified semaphore is 0, the resource count remains unchanged at 0, the invoking task is placed in the queue, and is moved to the waiting state to acquire the semaphore resource.

pol_sem is the polling system call execute the process of wai_sem, twai_sem is the system call as wai_sem with an additional timeout feature. Moreover, tmout can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, twai_sem that set tmout to TMO_FEVR will be used as wai_sem, twai_sem that set tmout to TMO_POL will be used as pol_sem

ref_sem Reference semaphore state

【Format】

ER ercd = ref_sem(ID semid, T_RSEM *pk_rsem) ;

【Parameter】

ID	semid	ID number of the semaphore to be referenced
T_RSEM*	pk_rsem	Pointer to the packet returning the semaphore state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rsem includes (T_RSEM type)

ID	wtskid	ID number of task at the head of the semaphore's queue
UINT	semcnt	Current semaphore resource count

【Error code】

E_ID	Invalid ID number (semid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified semaphore is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the semaphore with an ID number specified by returns the state through the packet specified by pk_rsem.

The ID number of the task at the head of the semaphore's queue is returned through wtskid. If no tasks are waiting to acquire the resource, TSK_NONE (=0) is returned.

The current resource count of the specified semaphore is returned through semcnt.

5.4.2 Eventflag

cre_flg	**Create eventflag**
acre_flg	**Create eventflag (Automatic ID assignment)**

【Format】

ER ercd = cre_flg(ID flgid, T_CFLG*pk_cflg) ;
ER_ID flgid = acre_flg(T_CFLG*pk_cflg) ;

【Parameter】

ID	flgid	ID number of the eventflag to be created (except acre_flg)
T_CFLG *	pk_cflg	Pointer to the packet containing the eventflag creation information)

pk_cflg includes (T_CFLG type)

ATR	flgatr	Eventflag attribute
FLGPTN	iflgptn	Initial value of eventflag bit pattern
VB const *	name	Eventflag name (character string)

【Return value】

In case of cre_flg

ER	ercd	Successful completion (E_OK) or error code

In case of acre_flg

ER_ID	flgid	ID number (positive value) of the created eventflag or error code

【Error code】

E_RSATR	Reserved attribute (flgatr is invalid or unusable)
E_PAR	Parameter error (iflgptn is invalid)
E_ID	Invalid ID number (flgid is invalid or unusable ; only cre_flg)
E_CTX	Context error (invoked from others except task and interrupt handler)
E_NOMEM	Insufficient memory (control area can not be allocated)
E_NOID	Lack of ID number (there is no eventflag ID assignable ; only acre_flg)
E_OBJ	Object state error (specified eventflag is already registered ; only cre_flg)

【Invoking context】	**cre_flg**	**acre_flg**
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

This system call creates the event flag with an ID number specified by flgid based on the eventflag creation information specified by pk_cflg. flgatr is the attribute of the eventflag, iflgptn is the initial value of the bit pattern after the event flag creation.

acre_flg assigns an ID number with biggest value from the pool of unassigned eventflag IDs for the evenflag to be created and returns the assigned ID number as a return value.

flgatr can be specified as ((TA_TFIFO || TA_TPRI) | (TA_WSGL || TA_WMUL) | [TA_CLR]). The eventflag's queue will be in FIFO order, if TA_TFIFO (＝0x00) is specified, the eventflag's queue will be in task priority order if TA_TPRI (＝0x01) is specified.

If TA_WSGL (=0x00) is specified, multiple tasks can not enter into the WATITING state for one event flag at the same time. While if TA_WMUL(=0x02) is specified , multiple tasks are possible to be in the WAITING state at the same.

If TA_CLR(=0x04) is specified, the task is released from the waiting for the eventflag when the release condition of eventflag waiting has been established, and the same time, the eventflag's entire bit pattern will be cleared.

del_flg Delete eventflag

【Format】

ER ercd = del_flg(ID flgid) ;

【Parameter】

ID	flgid	ID number of the eventflag to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (flgid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified eventflag is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the eventflag with an ID number specified by flgid.

set_flg	Set eventflag
iset_flg	

【Format】

ER ercd = set_flg(ID flgid, FLGPTN setptn) ;

ER ercd = iset_flg(ID flgid, FLGPTN setptn) ;

【Parameter】

ID	flgid	ID number of the eventflag to be set
FLGPTN	setptn	Bit pattern to set

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (setptn is invalid)
E_ID	Invalid ID number (flgid is invalid, or unusable)
E_NOMEM	Insufficient memory (lack of SSB : only for invoking from the interrupt handler)
E_NOEXS	Non-existent object (specified eventflag is not registered)

【Invoking context】	set_flg	iset_flg
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

This system call updates the bit pattern of the eventflag with an ID number specified by flgid to the bit-wise OR of setptn value and the bit pattern before calling the system call. When bit pattern of the eventflag is updated, each task in the eventflag's queue is checked starting from the head whether if the task satisfies the waiting release condition. If there is task that satisfies its waiting release conditions, that task is released from waiting. Besides, the released task receives E_OK as a return value of the system call that placed the task in the WAITING state. It also receives the bit pattern at this time as the bit pattern when releasing from waiting. At this time, if TA_CLR attribute is specified in specified eventflag's attribute, the system call complete after clearing the entire bit pattern of the evenflag. If TA_CLR attribute is not specified, keep checking the waiting queue to the end of the queue.

【Recommendation】

set_flg and iset_flg in μC3/Standard are the same system call, and both can be used by the same method irregardless of the invoking context. However, it is recommended that set_flg is used when invoking from the task context and iset_flg should be used for the remains.

clr_flg Clear event flag

【Format】

ER ercd = clr_flg(ID flgid, FLGPTN clrptn) ;

【Parameter】

ID	flgid	ID number of the eventflag to be set (cleared)
FLGPTN	clrptn	Bit pattern to be cleared (bitwise inverted value)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (flgid is invalid, or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified eventflag is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call updates the bit pattern of the specified eventflag with an ID specified by flgid to the bitwise AND of clrptn value and the bit pattern before calling the system call.

【Usage】

```
ER ercd;
                    /* invoke the value that set for only the clear bit to 0 as a parameter */
ercd = clr_flg(ID_Flag1, ~0x00000001);   /* clear only bit 0 of the eventflag */
```

wai_flg	**Wait for eventflag**
pol_flg	**Wait for eventflag (Polling)**
twai_flg	**Wait for eventflag (with Timeout)**

【Format】

```
ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode,
                  FLGPTN   *p_flgptn) ;
ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode,
                  FLGPTN   *p_flgptn) ;
ER ercd = twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode,
                  FLGPTN   *pflgptn, TMO tmout) ;
```

【Parameter】

ID	flgid	ID number of the eventflag to wait for
FLGPTN	waiptn	Wait bit pattern
MODE	wfmode	Wait mode
TMO	tmout	Specified timeout (only twai_flg)

【Return value】

ER	ercd	Successful completion (E_OK) or error code
FLGPTN	flgptn	Bit pattern when releasing from waiting

【Error code】

E_PAR	Parameter error (waiptn, wfmode is invalid)
E_ID	Invalid ID number (flgid is invalid, or unusable)
E_CTX	Context error (invoked from others except task ：except pol_flg, invoked from interrupt handler：only pol_flg, dispatch pending state：except pol_flg)
E_ILUSE	Illegal system call use (there is already a task waiting for an evenflag with the TA_WSGL attribute)
E_NOEXS	Non-existent object (specified eventflag is not registered)
E_RLWAI	Forced release from waiting state (rel_wai is accepted while waiting ; except pol_flg)
E_TMOUT	Polling failure or timeout (except wai_flg)
E_DLT	Waiting object deleted (event flag is deleted while waiting；except pol_flg)

【Invoking context】	wai_flg	pol_flg	twai_flg
Task	Possible	Possible	Possible
Initialization handler	Impossible	Possible	Impossible

Time event handler	Impossible	Possible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

This system call places the invoking task in the wait queue and moves to the eventflag waiting state until the eventflag satisfies the release condition in case that the bit pattern of the eventflag with an ID number specified by flgid are not satisfied their release condition specified by waiptn and wfmode. If the release condition specified by waiptn and wfmode is already satisfied, the system call complete without causing the invoking task to wait, the bit pattern which satisfied the release condition is returned through flgptn. At this time, if TA_CLR attribute is specified in the specified eventflag's attribute, all the bits in the eventflag's bit pattern are cleared.

When TA_WSGL attribute is specified in the eventflag's attribute and another task is connected to the eventflag's queue, an E_ILUSE error is returned regardless of the waiting release condition.

wfmode can be specified as either TWF_ANDW or TWF_ORW. When wfmode has TWF_ANDW set, the waiting release condition is a condition that entire bits specified by waiptn and bit pattern of the specified eventflag are set. If TWF_ORW is specified, it is a condition when one of bits specified by waiptn and bit pattern of the specified eventflag are set.

pol_flg is the polling system call execute the process of wai_flg, twai_flg is the system call as wai_flg with an additional timeout feature. Moreover, tmout can be set to TMO_POL (=0) or TMO_FEVR (=-1). In μC3/Standard, twai_flg that set tmout to TMO_FEVR will be used as wai_flg, twai_flg that set tmout to TMO_POL will be used as pol_flg

【Usage】

```
FLGPTN waiptn;          /* the area that store the flag pattern is allocated */
ER ercd;
                        /* invoke the pointer to the stored area as a parameter */
ercd = wai_flg(ID_Flag1, 0x00000003, TWF_ORF, &waiptn);
if (ercd == E_OK) {
    ercd = clr_flg(ID_Flag1, ~waiptn);
    if ((waiptn & 0x00000001) != 0) {
        /* the process for eventflag bit 0 */
    }
    if ((waiptn & 0x00000002) != 0) {
        /* the process for eventflag bit 1 */
    }
}
```

ref_flg Reference eventflag state

【Format】

ER ercd = ref_fig(ID flgid, T_RFLG *pk_rflg) ;

【Parameter】

ID	flgid	ID number of the eventflag to be referenced
T_RFLG*	pk_rflg	Pointer to the packet returning the eventflag state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rflg includes (T_RFLG type)

ID	wtskid	ID number of the task at the head of the eventflag's queue
FLGPTN	flgptn	Eventflag's current bit pattern

【Error code】

E_ID	Invalid ID number (flgid is invalid, or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified eventflag is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the eventflag with an ID number specified by flgid, and the state is returned through the packet specified by pk_rflg.

The ID number of the task at the head of the specified eventflag's wait queue is returned through wtskid. In case of no tasks are waiting for the event, TSK_NONE (=0) is returned.

The current bit pattern of the specified eventflag is returned through flgptn.

5.4.3 Data queues

cre_dtq	**Create data queue**
acre_dtq	**Create data queue (Automatic ID assignment)**

【Format】

ER ercd = cre_dtq(ID dtqid, T_CDTQ*pk_cdtq) ;
ER_ID dtqid= acre_dtq(T_CDTQ*pk_cdtq) ;

【Parameter】

ID	dtqid	ID number of the data queue to be created (except acre_dtq)
T_CDTQ*	pk_cdtq	Pointer to the packet containing the data queue creation information

pk_cdtq includes (T_CDTQ type)

ATR	dtqatr	data queue attribute
UINT	dtqcnt	Capacity of the data queue area (the number of data)
VP	dtq	Start address of the data queue
VB const *	name	Data queue name (character string)

【Return value】

In case of cre_dtq

ER	ercd	Successful completion (E_OK) or error code

In case of acre_dtq

ER_ID	dtqid	ID number (positive value) of the created data queue or error code

【Error code】

E_RSATR	Reserved attribute (dtqatr is invalid or unusable)
E_ID	Invalid ID number (dtqid is invalid or unusable ; only cre_dtq)
E_CTX	Context error (invoked from others except task or initialization handler)
E_NOMEM	Insufficient memory (management area and data queue area can not be allocated)
E_NOID	Lack of ID number (there is no data queue ID assignable ; only acre_dtq)
E_OBJ	Object state error (specified data queue is already registered ; only cre_dtq)

【Invoking context】	cre_dtq	acre_dtq
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

These system call creates the data queue with an ID number specified by dtqid based on the data queue creation information specified by pk_cdtq. dtqatr is the attribute of the data queue, dtqcnt is the number of datas that can be stored in the data queue area, dtq is the start address of the data queue area.

acre_dtq assigns an ID number with biggest value from the pool of unassigned data queue IDs for the data queue to be created and returns the assigned ID number as a return value.

dtqatr can be specified as (TA_TFIFO || TA_TPRI). If TA_TFIFO (0x00) is specified, the send-wait queue of the data queue will be in the FIFO order, if TA_TPRI (0x01) is specified, the send-wait queue of the data queue will be in task priority order.

The necessary area to store dtqcnt data starts from the address specified by dtq is used as the data queue area. If dtq is specified as NULL (= 0), the necessary memory area is automatically allocated from the memory area used for defined memory pool by the configuration.

If the synchronous message function is used, dtqcnt is specified as 0.

del_dtq Delete data queue

【Format】

ER ercd = del_dtq(ID dtqid) ;

【Parameter】

ID	dtqid	ID number of the data queue to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (dtqid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified data queue is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the data queue with an ID number specified by dtqid.

snd_dtq	**Send to data queue**
psnd_dtq	**Send to data queue (Polling)**
ipsnd_dtq	
tsnd_dtq	**Send to data queue (with Timeout)**

【Format】

ER ercd = snd_dtq(ID dtqid, VP_INT data) ;
ER ercd = psnd_dtq(ID dtqid, VP_INT data) ;
ER ercd = ipsnd_dtq(ID dtqid, VP_INT data) ;
ER ercd = tsnd_dtq(ID dtqid, VP_INT data, TMO tmout)

【Parameter】

ID	dtqid	ID number of the data queue to which the data is sent
VP_INT	data	Data to be sent to the data queue
TMO	tmout	Specified timeout (only tsnd_dtq)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (dtqid is invalid or unusable)
E_CTX	Context error (invoked from others except task : except (i)psnd_dtq, invoked from the interrupt handler : only (i)psnd_dtq, dispatch pending state : except (i)psnd_dtq)
E_NOMEM	Insufficient memory (lack of SSB: only for invoking from the interrupt handler)
E_NOEXS	Non-existent object (specified data queue is not registered)
E_RLWAI	Forced release from waiting state (rel_wai is accepted while waiting ; only snd_dtq, tsnd_dtq)
E_TMOUT	Polling failure or timeout (except snd_dtq)
E_DLT	Wait object deleted (specified data queue is deleted while waiting ; except (i)psnd_dtq)

【Invoking context】	snd_dtq	psnd_dtq	ipsnd_dtq	tsnd_dtq
Task	Possible	Possible	Possible	Possible
Initialization handler	Impossible	Possible	Possible	Impossible
Time event handler	Impossible	Possible	Possible	Impossible
Interrupt handler	Impossible	Possible	Possible	Impossible

【Description】

This system call sends the data to the task at the head of the receive-wait queue if there are already tasks waiting for receiving in the data queue specified by dtqid,and release the task from waiting. Besides, the released task receives E_OK as a return value of the system call that placed the task in WAITING state. It also receives the data value as data received from the data queue. In case of no tasks are waiting for receiving, the system call places the data to be sent at the tail of the data queue. If there is no space in the data queue area, the invoking task is placed in the send-wait queue and is moved to the sending waiting state for the data queue.

If no tasks are waiting for receiving in the specified data queue and no space in the data queue area, psnd_dtq and ipsnd_dtq return an E_TMOUT error as a return value. Nevertheless, if this system call is invoked from the interrupt handler and has its execution delayed, an E_TMOUT error can not be detected in the error code of the return values.

psnd_dtq and ipsnd_dtq is the polling system call execute the process of snd_dtq, tsnd_dtq is the system call as snd_dtq with an additional timeout feature. Moreover, tmout can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, tsnd_dtq that set tmout to TMO_FEVR will be used as snd_dtq, tsnd_dtq that set tmout to TMO_POL will be used as psnd_dtq.

【Recommendation】

psnd_dtq and ipsnd_dtq in μC3/Standard are the same system call, thus, both can be used by the same method regardless of the invoking context. However, it is recommended that psnd_dtq is used when invoking from the task context and ipsnd_dtq should be used for the remains.

fsnd_dtq	Forced send to data queue
ifsnd_dtq	

【Format】

ER ercd = fsnd_dtq(ID dtqid, VP_INT data) ;

ER ercd = ifsnd_dtq(ID dtqid, VP_INT data) ;

【Parameter】

ID	dtqid	ID number of the data queue to which the data is sent
VP_INT	data	Data to be sent to the data queue

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (dtqid is invalid, or unusable)
E_ILUSE	Illegal system use (capacity of the specified data queue area is 0)
E_NOMEM	Insufficient memory (lack of SSB: only for invoking from the interrupt handler)
E_NOEXS	Non-existent object (specified data queue is not registered)

【Invoking context】	fsnd_dtq	ifsnd_dtq
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

This system call send the data to the task at the head of the receive-wait queue if there are already tasks waiting for receiving in the data queue with an ID number specified by dtqid, and then release that task from waiting. Besides, the released task receives E_OK as a return value of the system call that placed the task in the WAITING state. It also receives the data value as data received from the data queue. If no tasks are waiting for receiving, the system call places the data to be sent at the tail of the data queue. Here, if here is no space in the data queue area, the system call deletes the first data of the data queue, reserve the necessary area in the data queue, and place the data to be sent at the tail of the data queue. In other words, the oldest data is deleted. In case the system call tries to forcibly send data to the data queue in which capacity of the data queue area is 0, an E_ILUSE error is returned as a return value.

【Recommendation】

fsnd_dtq and ifsnd_dtq in µC3/Standard are the same system calls, thus, both can be used by the same method irregardless of the invoking context. However, it is recommended that fsnd_dtq is used when invoking from the task context and ifsnd_dtq should be used for the remains.

rcv_dtq	**Receive from data queue**
prcv_dtq	**Receive from data queue (Polling)**
trcv_dtq	**Receive from data queue (with Timeout)**

【Format】

```
ER ercd = rcv_dtq(ID dtqid, VP_INT    *p_data) ;
ER ercd = prcv_dtq(ID dtqid, VP_INT    *p_data) ;
ER ercd = trcv_dtq(ID dtqid, VP_INT    *p_data, TMO tmout) ;
```

【Parameter】

ID	dtqid	ID number of the data queue from which a data is received
TMO	tmout	Specified timeout (only trcv_dtq)

【Return value】

ER	ercd	Successful completion (E_OK) or error code
VP_INT	data	Data received from the data queue

【Error code】

E_ID	Invalid ID number (dtqid is invalid or unusable)
E_CTX	Context error (invoked from others except task : except prcv_dtq, invoked from the interrupt handler : only prcv_dtq, dispatch pending state : except prcv_dtq)
E_NOEXS	Non-existent object (specified data queue is not regisered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting; except prcv_dtq)
E_TMOUT	Pollin failure or timeout (except rcv_dtq)
E_DLT	Waiting object deleted (specified data queue is deleted while waiting ; except prcv_dtq)

【Invoking context】	rcv_dtq	prcv_dtq	trcv_dtq
Task	Possible	Possible	Possible
Initialization handler	Impossible	Possible	Impossible
Time event handler	Impossible	Possible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

This system call removes the first data from the data queue with an ID number specified by dtqid if there are data placed in that queue, and returns the removed data through data. At this time, if there is a task waiting for sending in the data queue, the system call places the data

from the first task in the send-wait queue at the tail of the data queue and release that task from waiting. Besides, the released task receives E_OK as a return value of the system call that placed the task in the WAITING state.

If there are no data in the data queue and if there are tasks waiting to send in the specified data queue, the system call receives the data from the first task in the send-wait queue and release that task from the WAITING state. Besides, the released task receives E_OK as a return value of the system call that placed the task in the WAITING state. It also receives the accepted data through data.

In case that there are neither data nor tasks waiting to send, the invoking task is placed in the receive-wait queue and moved to the receiving waiting state for the data queue.

prcv_dtq is the polling system call execute the process of rcv_dtq, trcv_dtq is the system call as rcv_dtq with an additional timeout feature. Moreover, tmout can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, trcv_dtq that set tmout to TMO_FEVR will be used as rcv_dtq, trcv_dtq that set tmout to TMO_POL will be used as prcv_dtq.

ref_dtq Reference data queue state

【Format】

ER ercd = ref_dtq(ID dtqid, T_RDTQ *pk_rdtq) ;

【Parameter】

ID	dtqid	ID number of the data queue to be referenced
T_RDTQ*	pk_rdtq	Pointer to the packet returning the data queue state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rdtq includes (T_RDTQ type)

ID	stskid	ID number of the first task in the send-wait queue
ID	rtskid	ID number of the first task in the receive-wait queue
UINT	sdtqcnt	The number of data in the data queue

【Error code】

E_ID	Invalid ID number (dtqid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified data queue is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the data queue with an ID number specified by dtqid, and returns the state to the packet specified by pk_rdtq.

The ID number of the task at the head of the send-wait queue of the specified data queue is returned through stskid. If no tasks are waiting for sending, TSK_NONE (=0) is returned.

The ID number of the task at the head of the receive-wait queue of the specified data queue is returned through rtskid. If no tasks are waiting for receiving, TSK_NONE (=0) is returned.

The number of data currently in the data queue is returned through sdqcnt.

5.4.4 Mailboxes

cre_mbx	**Create mailbox**
acre_mbx	**Create mailbox (Automatic ID assignment)**

【Format】

ER ercd = cre_mbx(ID mbxid, T_CMBX *pk_cmbx) ;

ER_ID mbxid = acre_mbx(T_CMBX *pk_cmbx) ;

【Parameter】

ID	mbxid	ID number of the mailbox to be created (except acre_mbx)
T_CMBX*	pk_cmbx	Pointer to the packet containing the mailbox creation information

pk_cmbx includes (T_CMBX type)

ATR	mbxatr	Mailbox attribute
PRI	maxmpri	Maximum message priority of the message to be sent
VP	mprihd	Start address of the area for message queue headers for each message priority
VB const *	name	Mailbox name (character string)

【Return value】

In case of cre_mbx

ER	ercd	Successful completion (E_OK) or error code

In case of acre_mbx

ER_ID	mbxid	ID number (positive value) of the created mailbox Error code

【Error code】

E_RSATR	Reserved attribute (mbxatr is invalid or unusable)
E_PAR	Parameter error (maxmpri is invalid)
E_ID	Invalid ID number (mbxid is invalid or unusable ; only cre_mbx)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (control area can not be allocated)
E_NOID	Lack of ID number (there is no mailbox ID assignable ; only acre_mbx)

【Invoking context】	**cre_mbx**	**acre_mbx**
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible

Interrupt handler	Impossible	Impossible

【Description】

This system call creates the mailbox with an ID number specified by mbxid based on the mailbox creation information specified by pk_cmbx. mbxatr is the attribute of the mailbox, maxmpri is the maximum priority of messages sent to the mailbox, mprihd is the start address of the area for message queue headers for each mailbox priority.

maxmpri and mprihd are valid only when TA_MRI (＝0x02) is specified in mbxatr.

acre_mbx assigns an ID number with biggest value from the pool of unassigned mailbox IDs for the mailbox to be created and returns the assigned ID number as a return value.

mbxatr can be specified as ((TA_TFIFO || TA_TPRI)|(TA_MFIFO || TA_MPRI)). If TA_TFIFO (0x00) is specified, the mailbox's wait queue will be in the FIFO order, if TA_TPRI (0x01) is specified, the mailbox's wait queue will be in task priority order. Moreover, if TA_MFIFO (＝0x00) is specified, the message queue of the mailbox will be in the FIFO order, if TA_MPRI (＝0x02) is specified, the message queue of the mailbox will be in task priority order.

If TA_MPRI is specified in mbxatr, the necessary memory area when the maximum priority of the message to be sent is maxmpri starts from the address specified by maxmpri is used as the message queue area for each priority.

If mprihd is specified as NULL (= 0), the necessary memory area is automatically allocated from the memory area used for defined system by the configuration. Besides, if maxmpri is specified as 0 and a value greater than the maximum message priority (TMAX_MPRI), and E_PAR error is returned as a return value.

del_mbx Delete mailbox

【Format】

ER ercd = del_mbx(ID mbxid) ;

【Parameter】

ID	mbxid	ID number of the mailbox to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mbxid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified mailbox is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the mailbox with an ID number specified by mbxid.

snd_mbx Send to mailbox

【Format】

ER ercd = snd_mbx(ID mbxid, T_MSG *pk_msg) ;

【Parameter】

ID	mbxid	ID number of the mailbox to which the message is sent
T_MSG *	pk_msg	Start address of the message packet to be sent to the mailbox

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mbxid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified mailbox is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call sends the start address of the message packet specified by pk_msg to the task at the head of the wait queue if there is task waiting for receiving in the mailbox with an ID number specified by mbxid, and then releases that task from waiting. Besides, the released task receives E_OK as a return value of the system call that placed the task in the WAITING state and it also receives the value of pk_msg as the start address of the message packet received from the mailbox.

If no tasks are waiting for receiving, the system call places the message packet whose start address is specified by pk_msg in the message queue. If the message queue is in FIFO order, the message packet is placed at the tail of the message queue, and in case of the message priority order, the message packet is placed in the its message priority order. At this time, if there are messages with the same priority, the message is placed after those messages.

【Usage】

In case that the message in the FIFO order is sent

```
T_MSGPKT* pk_msgpkt;   /* the area that store the start address of the message packet is allocated */
ER ercd;
ercd = get_mpf(ID_Mpf1, &pk_msgpkt);
if (ercd == E_OK) {
        /* Edit message packet */
        /* invoke the pointer to the stored area as a parameter*/
    ercd = snd_mbx(ID_Mbx1, (T_MSG*)pk_msgpkt);
}
```

In case that the message in the message priority order is sent

```
T_MSGPRIPKT* pk_msgpripkt;
ER ercd;
ercd = get_mpf(ID_Mpf1, &pk_msgpripkt);
if (ercd == E_OK) {
        /* edit the message packet */
        /* set the message priority to 1 */
    pk_msgpripkt->pk_msg.msgpri = 1;
        /* invoke the pointer to the stored area as a parameter */
    ercd = snd_mbx(ID_Mbx1, (T_MSG*)pk_msgpkt);
}
```

rcv_mbx	**Receive from mailbox**
prcv_mbx	**Receive from mailbox (Polling)**
trcv_mbx	**Receive from mailbox (with Timeout)**

【Format】

```
ER ercd = rcv_mbx(ID mbxid, T_MSG    **ppk_msg) ;
ER ercd = prcv_mbx(ID mbxid, T_MSG    **ppk_msg) ;
ER ercd = trcv_mbx(ID mbxid, T_MSG    **ppk_msg, TMO tmout) ;
```

【Parameter】

ID	mbxid	ID number of the mailbox from which a message is received
TMO	tmout	Specified timeout (only trcv_mbx)

【Return value】

ER	ercd	Successful completion (E_OK) or error code
T_MSG*	pk_msg	Start address of the message packet received form the mailbox

【Error code】

E_ID	Invalid ID number (mbxid is invalid or unusable)
E_CTX	Context error (invoked from others except task : except prcv_mbx, invoked from the interrupt handler : only prcv_mbx, dispatch pending state : exept prcv_mbx)
E_NOEXS	Non-existent object (specified mailbox is not registered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting; except prcv_mbx)
E_TMOUT	Polling failure or timeout (except rcv_mbx)
E_DLT	Waiting object deleted (specified mailbox is deleted while waiting ; except prcv_mbx)

【Invoking context】	rcv_mbx	prcv_mbx	trcv_mbx
Task	Possible	Possible	Possible
Initialization handler	Impossible	Possible	Impossible
Time event handler	Impossible	Possible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

This system call removes the first message packet from the message queue of the mailbox with an ID number specified by mbxid if there is message placed in the message queue and

returns its start address through pk_msg. If there are no messages in the message queue, the invoking task is placed in the wait queue and moved to the receiving waiting state for the mailbox.

prcv_mbx is the polling system call execute the process of rcv_mbx, trcv_mbx is the system call as rcv_mbx with an additional timeout feature. Moreover, tmout can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, trcv_mbx that set tmout to TMO_FEVR will be used as rcv_mbx, trcv_mbx that set tmout to TMO_POL will be used as prcv_mbx.

【Usage】

```
T_MSGPKT* pk_msgpkt;   /* the area that store the start address of the message packet is allocated */
ER ercd;

ercd = rcv_mbx(ID_Mbx1, (T_MSG    **)&pk_msgpkt);
if (ercd == E_OK) {
                    /* process the message packet */
                    /* if the fixed-length memory pool is used, the memory block is released */
    ercd = rel_mpf(ID_mpf1, pk_msgpkt);
}
```

ref_mbx Reference mailbox state

【Format】

ER ercd = ref_mbx(ID mbxid, T_RMBX *pk_rmbx) ;

【Parameter】

ID	mbxid	ID number of the mailbox to be referenced
T_RMBX *	pk_rmbx	Pointer to the packet returning the mailbox state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rmbx includes (T_RMBX type)

ID	wtskid	ID number of the first task in the mailbox's wait queue
T_MSG *	pk_msg	Start address of the first message packet of the message queue

【Error code】

E_ID	Invalid ID number (mbxid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified mailbox is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the mailbox with an ID number specified by mbxid, and returns the state through the packet specified by pk_rmbx.

The ID number of the first task of the specified mailbox's wait queue is returned through wtskid. If no tasks are waiting for receiving, TSK_NONE (=0) is returned.

The start address of the first message packet of the specified mailbox's message queue is returned through pk_msg. If there is no message in the message queue, NULL (=0) is returned.

5.4.5 Mutexes

cre_mtx	**Create mutex**
acre_mtx	**Create mutex (Automatic ID assignment)**

【Format】

ER ercd = cre_mtx(ID mtxid, T_CMTX*pk_cmtx) ;
ER_ID mtxid = acre_mtx(T_CMTX*pk_cmtx) ;

【Parameter】

ID	mtxid	ID number of the mutex to be created (except acre_mtx)
T_CMTX*	pk_cmtx	Pointer to the packet containing the mutex creation information

pk_cmtx includes (T_CMTX type)

ATR	mtxatr	Mutex attribute
PRI	ceilpri	Mutex ceiling priority
VB const *	name	Mutex name (charactering string)

【Return value】

In case of cre_mtx

ER	ercd	Successful completion (E_OK) or error code

In case of acre_mtx

ER_ID	mtxid	ID number (positive value) of the created mutex or Error code

【Error code】

E_RSATR	Reserved attribute (mtxatr is invalid or unusable)
E_PAR	Parameter error (ceilpri is invalid)
E_ID	Invalid ID number (mtxid is invalid or unusable ; only cre_mtx)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (control area can not be allocated)
E_NOID	Lack of ID number (there is no mutex ID assignable ; only acre_mtx)
E_OBJ	Object state error (specified mutex is already registered ;only cre_mtx)

【Invoking context】	cre_mtx	acre_mtx
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

This system call creates a mutex with an ID number specified by mtxid based on the mutex creation information specified by pk_cmtx. mtxatr is the attribute of the mutex. ceilpri is the mutex ceiling priority. ceilpri is only valid when mtxatr has TA_UPPER LIMIT OF (= 0x03) set.

acre_mtx assigns an ID number with biggest value from the pool of unassigned mutex IDs for the mutex to be created and returns the assigned ID number as a return value.

mtxatr can be specified as (TA_TFIFO || TA_TPRI || TA_INHERIT || TA_CEILING). If TA_TFIFO (= 0x00) is specified, the mutex's wait queue will be in FIFO order. Otherwise, the mutex's wait queue will be in task priority order. Besides, in case TA_INHERIT (= 0x02) is set, the current priority of a task is controlled according to the priority inheritance protocol, if TA_CEILING (= 0x03) is set, the current priority of a task is controlled according to the priority ceiling protocol.

del_mtx Delete mutex

【Format】

ER ercd = del_mtx(ID mtxid) ;

【Parameter】

ID	mtxid	ID number of the mutex to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mtxid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified mutex is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the mutex with an ID number specified by mtxid.

loc_mtx	**Lock mutex**
ploc_mtx	**Lock mutex (Polling)**
tloc_mtx	**Lock mutex (with Timeout)**

【Format】

ER ercd = loc_mtx(ID mtxid) ;

ER ercd = ploc_mtx(ID mtxid) ;

ER ercd = tloc_mtx(ID mtxid, TMO tmout) ;

【Parameter】

ID	mtxid	ID number of the mutex to be locked
TMO	tmout	Specified timeout (only tloc_mtx)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mtxid is invalid or unusable)
E_CTX	Context error (invoked from others except task and initialization handler)
E_ILUSE	Illegal service call use (multiple locking of a mutex, or ceiling priority violation)
E_NOEXS	Non-existent object (specified mutex is not registered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting; except ploc_mtx)
E_TMOUT	Polling failure or timeout (except loc_mtx))
E_DLT	Waiting object deleted (mutex is deleted while waiting; except ploc_mtx))

【Invoking context】	loc_mtx	ploc_mtx	tloc_mtx
Task	Possible	Possible	Possible
Initialization handler	Impossible	Impossible	Impossible
Time event handler	Impossible	Impossible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

This system call locks the mutex with an ID number specified by mtxid. In other words, if the specified mutex is not locked, the system call let the invoking task locks the mutex. If the specified mutex is locked, the invoking task is placed in the wait queue and moved to the waiting state to lock the mutex.

In case that the current priority of the invoking task does not match the priority control rule of

μC3/Standard caused by the specified mutex is locked, the current priority of tasks (including invoking tasks exception) is changed appropriately.

If the invoking task has already locked the specified mutex, an E_ILUSE error is returned as a return value. Moreover, if the specified mutex has the TA_CEILING attribute set and the invoking task has a base priority higher than the ceiling priority of the specified mutex, an E_ILUSE error is returned as a return value.

ploc_mtx is the polling system call execute the process of loc_mtx, tloc_mtx is the system call as loc_mtx with an additional timeout feature. Moreover, tmout can be set to a positive number indicating a timeout duration or it can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, tloc_mtx that set tmout to TMO_FEVR will be used as loc_mtx, tloc_mtx that set tmout to TMO_POL will be used as ploc_mtx

unl_mtx Unlock mutex

【Format】

ER ercd = unl_mtx(ID mtxid) ;

【Parameter】

ID	mtxid	ID number of the mutex to be unlocked

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mtxid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_ILUSE	Illegal service call use (specified mutex is not locked)
E_NOEXS	Non-existent object (specified mutex is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call unlocks the mutex with an ID number specified by mtxid. In other words, if there are tasks waiting to lock the specified mutex, the system call releases the first task in the wait queue from waiting and lets the released task lock the mutex. At this time, the released task receives E_OK as a return value of the system call that placed the task in the WAITING state. If no task is waiting to lock, the system call moves the specified mutex to the unlocked state.

In case that the current priority of the invoking task does not match the priority control rule of μC3/Standard caused by the mutex is unlocked, the current priority of tasks (including non-invoking tasks) is changed appropriately

When the invoking task does not have the mutex locked, the system call returns an E_ILUSE error as a return value.

ref_mtx Reference mutex state

【Format】

ER ercd = ref_mtx(ID mtxid, T_RMTX*pk_rmtx) ;

【Parameter】

ID	mtxid	ID number of the mutex to be referenced
T_RMTX*	pk_rmtx	Pointer to the packet returning the mutex state

【Return value】

ER	ercd	Successful completion (E_OK) or error code
pk_rmtx includes (T_RMTX type)		
ID	htskid	ID number of the task locking the mutex
ID	wtskid	ID number of the first task in the mutex's wait queue

【Error code】

E_ID	Invalid ID number (mtxid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified mutex is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the mutex with an ID number specified by mtxid and returns the state through the packet specified by pk_rmtx.

The ID number of the task that has the specified mutex locked is returned through htskid. If the specified mutex is not locked, TSK_NONE (= 0) is returned.

The ID number of the task at the head of the specified mutex's wait queue is returned through wtskid. If no tasks are waiting to lock the mutex TSK_NONE (= 0) is returned.

5.4.6 Message buffers

cre_mbf	**Create message buffer**
acre_mbf	**Create message buffer (Automatic ID assignment)**

【Format】

ER ercd = cre_mbf(ID mbfid, T_CMBF * pk_cmbf) ;

ER_ID mbfid = acre_mbf(T_CMBF * pk_cmbf) ;

【Parameter】

ID	mbfid	ID number of the message buffer to be created (except acre_mbf)
T_CMBF*	pk_cmbf	Pointer to the packet containing the message buffer creation information

pk_cmbf includes (T_CMBF type)

ATR	mbfatr	Message buffer attribute
UINT	maxmsz	Maximum message size (in bytes)
SIZE	mbfsz	Size of message buffer area (in bytes)
VP	mbf	Start address of message buffer area
VB const *	name	Message buffer name (character string)

【Return value】

In case of cre_mbf

ER	ercd	Successful completion (E_OK) or error code

In case of acre_mbf

ER_ID	mbfid	ID number (positive value) of the created message buffer or error code

【Error code】

E_RSATR	Reserved attribute (mbfatr is invalid or unusable)
E_PAR	Parameter error (maxmsz or mbfsz is invalid)
E_ID	Invalid ID number (mbfid is invalid or unusable; only cre_mbf)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (management area and message buffer area can not be allocated)
E_NOID	Lack of ID number (there is no message buffer ID assignable; only acre_mbf)
E_OBJ	Object state error (specified message buffer is already register ; only

cre_mbf)

【Invoking context】	cre_mbf	acre_mbf
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

This system call creates a message buffer with an ID number specified by mbfid based on the message buffer creation information specified by pk_cmbf. mbfatr is the attribute of the message buffer. maxmsz is the maximum size (in bytes) of the message that can be sent to the message buffer. mbfsz is the size of the message buffer area (in bytes). mbf is the start address of the message buffer area.

acre_mbf assigns an ID number with biggest value from the pool of unassigned message buffer IDs for the message buffer to be created and returns the assigned ID number as a return value.

mbfatr can be specified as (TA_TFIFO || TA_TPRI). If TA_TFIFO (= 0x00) is specified, the message buffer's send-wait queue will be in FIFO order. If TA_TPRI (= 0x01) is specified, the message buffer's send-wait queue will be in task priority order.

The memory area with a size of mbfsz bytes starting from the address specified by mbf is used as the message buffer area. Because the information for message management is also placed in the message buffer area, the whole message buffer area cannot be used to store messages. An application program can estimate the size to be specified in mbfsz by using the TSZ_MBF. If mbf is specified as NULL (= 0), the memory area with size specified by mbfsz is allocated automatically from the memory area used for defined memory pool at the time of configuration. mbfsz may be specified as 0 when the synchronous message function is used.

When maxmsz is specified as 0 or a value greater than 65535, an E_PAR error is returned as a return value.

del_mbf Delete message buffer

【Format】

ER ercd = del_mbf(ID mbfid) ;

【Parameter】

ID	mbfid	ID number of the message buffer to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mbfid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified message buffer is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the message buffer with an ID number specified by mbfid.

snd_mbf	**Send to Message buffer**
psnd_mbf	**Send to Message buffer (Polling)**
tsnd_mbf	**Send to Message buffer (with Timeout)**

【Format】

ER ercd = snd_mbf(ID mbfid, VP msg, UINT msgsz) ;
ER ercd = psnd_mbf(ID mbfid, VP msg, UINT msgsz) ;
ER ercd = tsnd_mbf(ID mbfid, VP msg, UINT msgsz, TMO tmout) ;

【Parameter】

ID	mbfid	ID number of the message buffer to which the message is sent
VP	msg	Start address of the message to be sent
UINT	msgsz	Size of the message to be sent (in bytes)
TMO	tmout	Specified time-out (only tsnd_mbf)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (msg, msgsz, tmout is invalid)
E_ID	Invalid ID number (mbfid is invalid or unusable)
E_CTX	Context error (invoked from others except task :except psnd_mbf, invoked from interrupt handler : only psnd_mbf, dispatch pending state :except psnd_mbf)
E_NOEXS	Non-existent object (specified message buffer is not registered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting; except psnd_mbf)
E_TMOUT	Polling failure or timeout (except snd_mbf)
E_DLT	Waiting object deleted (specified message buffer is deleted while waiting; except psnd_mbf)

【Invoking context】	snd_mbf	psnd_mbf	tsnd_mbf
Task	Possible	Possible	Possible
Initialization handler	Impossible	Possible	Impossible
Time event handler	Impossible	Possible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

This system call sends a message stored in the memory area starting from the address specified by msg and whose size in bytes is specified by msgsz to the message buffer specified with an ID number specified by mbfid.

If there are tasks already waiting for receiving in the specified message buffer, the system call copy the sent message to the area in which the first task in the receive-wait queue store the received message and release the task from waiting. At that time, the released task receives the size of the sent message (msgsz) as the return value of the system call that placed the task in the WAITING state.

If no tasks are waiting for receiving in the specified message buffer, the behavior of the system depends on whether there is a task already waiting to send its message before the invoking task. This system call copies the sent message to the tail of the message buffer if either no task is waiting to send a message to the specified message buffer, or the priorities of the other tasks that are waiting to send messages are lower than the invoking task in the task priority order. If neither of these conditions is satisfied, or if there is no space in the message buffer area to store the sent message, the invoking task is placed in the send-wait queue and moved to the sending waiting state for the message buffer.

When msgsz is larger than the maximum message size of the message buffer, an E_PAR error is returned. Besides, an E_PAR error is also returned as a return value when msgsz is specified as 0.

psnd_mbf is the polling system call execute the process of snd_mbf, tsnd_mbf is the system call as snd_mbf with an additional timeout feature. Moreover, tmout can be set to a positive number indicating a timeout duration or it can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, tsnd_mbf that set tmout to TMO_FEVR will be used as snd_mbf, tsnd_mbf that set tmout to TMO_POL will be used as psnd_mbf

rcv_mbf	**Receive from Message buffer**
prcv_mbf	**Receive from Message buffer (Polling)**
trcv_mbf	**Receive from Message buffer (with Timeout)**

【Format】

```
ER_UINT msgsz = rcv_mbf(ID mbfid, VP msg) ;
ER_UINT msgsz = prcv_mbf(ID mbfid, VP msg) ;
ER_UINT msgsz = trcv_mbf(ID mbfid, VP msg, TMO tmout) ;
```

【Parameter】

ID	mbfid	ID number of the message buffer from which a message is received
VP	msg	Start address to store the received message
TMO	tmout	Specified timeout (only trcv_mbf)

【Return value】

ER_UINT	msgsz	Size of the received message (in bytes, positive value) or error code

【Error code】

E_ID	Invalid ID number (mbfid is invalid or unusable)
E_CTX	Context error (invoked from others except task：except prcv_mbf, invoked from the interrupt handler：only prcv_mbf, dispatch pending state: except prcv_mbf)
E_NOEXS	Non-existent object (specified message buffer is not registered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting；except prcv_mbf)
E_TMOUT	Polling failure or timeout (except rcv_mbf)
E_DLT	Waiting object deleted (specified message buffer is deleted while waiting；except prcv_mbf)

【Invoking context】	rcv_mbf	prcv_mbf	trcv_mbf
Task	Possible	Possible	Possible
Initialization handler	Impossible	Possible	Impossible
Time event handler	Impossible	Possible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

These system call receives a message from the message buffer with an ID number specified by mbfid, and stores that message from the address specified by msg. The byte count of the

received message will be returned through msgsz.

In case that the specified message buffer already has messages, the system call copies the first message to the memory area from the address specified by msg and returns the message size through msgsz. The copied message is deleted from the message buffer area. In case there are tasks waiting for sending in the message buffer, the system call checks if there is enough space for the message of the first task of the send-wait queue in the message buffer area caused by deleting the message, if possible, the system call copies the message to the tail of the message buffer, and then releases the task from waiting. At that time, the released task receives E_OK as a return value of the system call that placed the task in the WAITING state. Besides, in case of some tasks waiting to send still remain, the same actions must be repeated on the new task at the head of the send-wait queue.

If no messages are stored and if there are tasks waiting for sending in the specified message buffer, the message from the first task in the send-wait queue is copied under the address specified by msg and that task is removed from waiting. Besides, the size of the copied messages is returned through msgsz. The released task receives E_OK as a return value of the system call that placed the task in the WAITING state.

If no messages are stored and if there are not tasks waiting for sending, the invoking task is placed in the receive-wait queue and moved to the receiving waiting state for the message buffer.

prcv_mbf is the polling system call execute the process of rcv_mbf, trcv_mbf is the system call as rcv_mbf with an additional timeout feature. tmout can be set to a positive number indicating a timeout duration or it can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, trcv_mbf that set tmout to TMO_FEVR will be used as rcv_mbf, trcv_mbf that set tmout to TMO_POL will be used as prcv_mbf.

ref_mbf Reference message buffer state

【Format】

ER ercd = ref_mbf(ID mbfid, T_RMBF*pk_rmbf) ;

【Parameter】

ID	mbfid	ID number of the message buffer to be referenced
T_RMBF*	pk_rmbf	Pointer to the packet returning the message buffer state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rmbf includes (T_RMBF type)

ID	stskid	ID number of the first task in the send-wait queue of the message buffer
ID	rtskid	ID number of the first task in the receive-wait queue of the message buffer
UINT	smsgcnt	The number of messages in the message buffer
SIZE	fmbfsz	Size of free message buffer area (in bytes, without the minimum control areas)

【Error code】

E_ID	Invalid ID number (mbfid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified message buffer is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the message buffer with an ID number specified by mbfid and returns the state through the packet specified by pk_rmbf.

The ID number of the first task in the specified message buffer's send-wait queue is returned through stskid. If no tasks are waiting for sending, TSK_NONE (=0) is returned.

The ID number of the first task in the specified message buffer's receive-wait queue is returned through rtskid. If no tasks are waiting for receiving, TSK_NONE (=0) is returned.

The number of messages currently in the message buffer is returned through smsgcnt.

The size of free message buffer area (in bytes) is returned through fmbfsz.

5.4.7 Rendezvous

cre_por	**Create rendezvous port**
acre_por	**Create rendezvous port (Automatic ID assignment)**

【Format】

ER ercd = cre_por(ID porid, T_CPOR*pk_cpor) ;
ER_ID porid = acre_por(T_CPOR*pk_cpor) ;

【Parameter】

ID	porid	ID number of the rendezvous port to be created (except acre_por)
T_CPOR*	pk_cpor	Pointer to the packet containing the rendezvous port creation information.

pk_cpor includes (T_CPOR type)

ATR	poratr	Rendezvous attribute
UINT	maxcmsz	Maximum calling message size (in bytes)
UINT	maxrmsz	Maximum reply message size (in bytes)
VB const *	name	Rendezvous port name (character string)

【Return value】

In case of cre_por

ER	ercd	Successful completion (E_OK) or error code

In case of acre_por

ER_ID	porid	ID number (positive value) of the created rendezvous port or error code

【Error code】

E_RSATR	Reserved attribute (poratr is invalid or unusable)
E_PAR	Parameter error (maxcmsz, maxrmsz is invalid)
E_ID	Invalid ID number (porid is invalid or unusable ; only cre_por)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (control area can not be allocated)
E_NOID	Lack of ID number (there is no rendezvous port ID assignable ; only acre_por)
E_OBJ	Object state error (specified rendezvous port is already registered ; only cre_por)

【Invoking context】	cre_por	acre_por
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

This system call creates the rendezvous port with an ID number specified by porid based on the rendezvous port creation information specified by pk_cpor. poratr is the rendezvous port's attribute, maxcmsz is maximum size (in bytes) of the calling message and maxrmsz is maximum size (in bytes) of the reply message.

acre_por assigns an ID number with biggest value from the pool of unassigned rendezvous port IDs for the rendezvous port to be created and returns the assigned ID number as a return value.

poratr can be specified as (TA_TFIFO || TA_TPRI). If TA_TFIFO (＝0x00) is specified, the rendezvous port's call-wait queue will be in FIFO order, and if TA_TPRI (＝0x01) is specified, it will be in the task priority order.

When a value greater than 65535 is specified in maxcmsz or mazrmsz, an E_PAR error is returned as a return value. maxcmsz and mazrmsz can be specified as 0.

del_por Delete rendezvous port

【Format】

ER ercd = del_por(ID porid) ;

【Parameter】

ID	porid	ID number of the rendezvous port to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (porid is invalid or unusable)
E_CTX	Context error (invoked from the others except task)
E_NOEXS	Non-existent object (specified rendezvous port is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the rendezvous port with an ID number specified by porid.

cal_por	**Call rendezvous**
tcal_por	**Call rendezvous (with Timeout)**

【Format】

```
ER_UINT rmsgsz = cal_por(ID porid, RDVPTN calptn, VP msg, UINT cmsgsz) ;
ER_UINT rmsgsz = tcal_por(ID porid, RDVPTN calptn, VP msg,
                                                    UINT cmsgsz, TMO tmout) ;
```

【Parameter】

ID	porid	ID number of the rendezvous port to be called
RDVPTN	calptn	Bit pattern of the rendezvous condition on the calling side
VP	msg	Start address of the calling message or start address ／in which store the reply message
UINT	cmsgsz	Size of the calling message (in bytes)
TMO	tmout	Specified timeout (only tcal_por)

【Return value】

ER_UINT	rmsgsz	Size of the reply message (in bytes, positive value or 0) or error code

【Error code】

E_PAR	Parameter erro (calptn, cmsgsz, tmout is invalid)
E_ID	Invalid ID number (porid is invalid or unusable)
E_CTX	Context error (invoked from others except task, dispatch pending state)
E_NOEXS	Non-existent object (specified rendezvous port is not registered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting for calling)
E_TMOUT	Polling failure or timeout (only tcal_por)
E_DLT	Waiting object deleted (specified rendezvous port is deleted while waiting for rendezvous' s calling)

【Invoking context】	cal_por	tcal_por
Task	Possible	Possible
Initialization handler	Impossible	Impossible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

This system call calls a rendezvous for the rendezvous port with an ID number specified by porid with the rendezvous condition specified by calptn. The message stored in the memory area starting from the address specified by msg and whose size in bytes is specified by cmsgsz will be the calling message. Besides, the system call stores the reply message from the address specified by msg,and returns its size in bytes through rmsgsz.

When there is a task in the accepting waiting state for the rendezvous at the specified rendezvous port,if the rendezvous condition of that task and the rendezvous condition specified by calptn are established, the system call establishes a rendezvous. If there are more than one task in the rendezvous' s accepting waiting state, the system call checks their rendezvous conditions one by one starting from the first task in the accept-wait queue, then establishes a rendezvous with the first task that matches the rendezvous condition.

When a rendezvous is established, the system call assigns a rendezvous number in order to distinguish those rendezvous and moves the invoking task to the termination waiting state for the rendezvous. Besides, the system call also copies the calling message specified by msg and cmsgsz into the area in which the accepting task of the established rendezvous (task in accepting waiting state for the rendezvous) stores the calling message, and then releases the task from waiting. The released task receives the rendezvous number and the calling message size (cmsgsz) as the return value of the system call that placed the task in the WAITING state.

If no tasks are waiting to accept a rendezvous at the specified rendezvous port, or if the rendezvous condition is not established even if there exists task in the accepting waiting state for a rendezvous, the invoking task is placed in the call-wait queue and moved to the calling waiting state for the rendezvous.

tcal_por is the system call as cal_por with an additional timeout feature. tmout can be set to a positive number indicating a timeout duration or it can be set to TMO_FEVR (＝－1). If tmout is specified as TMO_POL (＝0), an E_PAR error is returned. In μC3/Standard, tcal_por that set tmout to TMO_POL will be used as cal_por

If tcal_por is invoked and results in a timeout after a rendezvous established, the state of the rendezvous can not be recovered to its former state before it was established. In this case, an error is reported to the rendezvous accepting task at the time that the task tries to complete the rendezvous. Besides, if rel_wai is invoked for the task in termination waiting state for a rendezvous, and the task is forcibly released from the termination waiting state for a rendezvous, there is different in returning of E_RLWAI error, but an error is reported at the time that the task tries to complete the rendezvous like the way of the rendezvous accepting task.

If calptn is specified as 0, an E_PAR error is returned as a return value. If cmsgsz is greater than the maximum calling message size of the specified rendezvous port, an E_PAR error is returned as a return value. cmsgsz can be specified as 0.

acp_por	**Accept rendezvous**
pacp_por	**Accept rendezvous (Polling)**
tacp_por	**Accept rendezvous (with timeout)**

【Format】

```
ER_UINT cmsgsz = acp_por(ID porid, RDVPTN acpptn,
                                    RDVNO*p_rdvno, VP msg) ;
ER_UINT cmsgsz = pacp_por(ID porid, RDVPTN acpptn,
                                    RDVNO*p_rdvno, VP msg) ;
ER_UINT cmsgsz = tacp_por(ID porid, RDVPTN acpptn,
                                    RDVNO*p_rdvno, VP msg, TMO tmout) ;
```

【Parameter】

ID	porid	ID number of the rendezvous port where a rendezvous is accepted
RDVPTN	acpptn	Bit pattern of the rendezvous condition on the accepting side.
VP	msg	Start address where store the calling message
TMO	tmout	Specified timeout (only tacp_por)

【Return value】

ER_UINT	cmsgsz	Size of the calling message (in bytes, positive value or 0) or error code
RDVNO	rdvno	Established rendezvous number

【Error code】

E_PAR	Parameter error (acpptn is invalid)
E_ID	Invalid ID number (porid is invalid or unusable)
E_CTX	Context error (invoked from others except task, dispatch pending state)
E_NOEXS	Non-existent object (specified rendezvous port is not registered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting ; except pacp_por)
E_TMOUT	Polling failure or timeout (except acp_por)
E_DLT	Waiting object deleted (specified rendezvous port is deleted while waiting ; except pacp_por)

【Invoking context】	acp_por	pacp_por	tacp_por
Task	Possible	Possible	Possible
Initialization handler	Impossible	Impossible	Impossible
Time event handler	Impossible	Impossible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

This system call accepts a rendezvous at the rendezvous port with an ID number specified by porid according to the rendezvous condition specified by acpptn. The calling message is stored starting from the address specified by msg and its size in bytes is returned through cmsgsz, the assigned rendezvous number is returned by rdvno.

If there is task in the calling waiting state for the rendezvous at the specified rendezvous port, plus the rendezvous condition specified by acpptn and the rendezvous condition of that task are established, the rendezvous is established. In case there are more than one task in the calling waiting state for the rendezvous**,** the system call checks their rendezvous conditions one by one starting from the first task in the call-wait queue, then establishes a rendezvous with the first task that matches the rendezvous condition.

When a rendezvous is established, the system call assigns a rendezvous number in order to distinguish those rendezvouses and returns it through rdvno. This system call also copies the calling message of the calling task of the established rendezvous (task in calling waiting state for the rendezvous) to the memory area staring from the address specified by msg, and then returns the message size through cmsgsz. The calling task of the rendezvous is removed from the rendezvous' s call-wait queue and moved to the termination waiting state for the rendezvous.

If no tasks are waiting to call a rendezvous at the specified rendezvous port, or if the rendezvous condition is not established even if there exists task in the calling waiting state for a rendezvous, the invoking task is placed in the accept-wait queue and moved to the accepting waiting state for the rendezvous.

pacp_por is the polling system call execute the process of acp_por, tacp_por is the system call as acp_por with an additional timeout feature. tmout can be set to a positive number indicating a timeout duration or it can be set to TMO_POL (＝0) and TMO_FEVR (＝－1).In μC3/Standard, tcal_por that set tmout to TMO_FEVR will be used as acp_por, tacp_por that set tmout to TMO_POL will be used as pacp_por.

If acpptn is specified as 0, an E_PAR error is returned as a return value.

fwd_por Forward rendezvous

【Format】

```
ER ercd = fwd_por(ID porid, RDVPTN calptn, RDVNO rdvno,
                                        VP msg, UINT cmsgsz) ;
```

【Parameter】

ID	porid	ID number of the rendezvous port of forwarding destination
RDVPTN	calptn	Bit pattern of the rendezvous condition on the calling side
RDVNO	rdvno	Rendezvous number to be forwarded
VP	msg	Calling message start address
UINT	cmsgsz	Size of the calling message (in bytes)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (calptn, cmsgsz is invalid)
E_ID	Invalid ID number (porid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_ILUSE	Illegal service call use (maximum reply message size of the rendezvous port to which the rendezvous is forwarded is too large)
E_OBJ	Object state error (rdvno is invalid)
E_NOEXS	Non-existent object (specified rendezvous port is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call forwards the rendezvous with the assigned rendezvous number specified by rdvno to the rendezvous port with an ID number specified by porid according to the rendezvous condition specified by calptn. The message with size in bytes is specified by cmsgsz which is stored in the memory area starting from the address specified by msg will be the calling message after forwarding. The others except the task accepting rendezvous also might forward the rendezvous by this system call.

If fwd_por is invoked, the result is the same as if the task that called the rendezvous specified by rdvno (hereinafter, it is called calling task) has called the rendezvous at the rendezvous port specified by porid according to the calling message and the rendezvous condition specified as a parameter of fwd_por.

If a task is waiting to accept a rendezvous at the rendezvous port of forwarding destination, and if the rendezvous condition of that task and the rendezvous condition specified by calptn are established, the system call establishes a rendezvous between the calling tasks. In case there are one more than task waiting to accept a rendezvous**,** the system call checks the rendezvous conditions one by one starting from the first task in the accept-wait queue, then establishes a rendezvous with the first task that matches the rendezvous condition.

If a rendezvous is established, the system call assigns a new rendezvous number and moves the calling task to the termination waiting state for the rendezvous at the new rendezvous port. Besides, the system call also copies the calling message specified by msg and cmsgsz into the area in which the accepting task of the established rendezvous (task in accepting waiting state for the rendezvous) stores the calling message, and then releases the task from waiting. The released task receives the new rendezvous number and the calling message size (cmsgsz) as the return value of the system call that placed the task in the WAITING state.

If no tasks are waiting to accept a rendezvous at the rendezvous port of forwarding destination, or if the rendezvous condition is not established even if there exists task in the accepting waiting state for a rendezvous, the calling task is placed in the call-wait queue of the rendezvous port of forwarding destination and moved to the calling waiting state for the rendezvous. At this time, the calling message specified by msg and cmsgsz is copied into the area where the calling task stores the reply message.

The maximum reply message size of the rendezvous port of forwarding destination must be smaller than the maximum reply size of the rendezvous port at which the forwarded rendezvous was established. If this condition is not satisfied, an E_ILUSE error is returned as a return value.

When cmsgsz is larger than the maximum calling message size of the rendezvous port of forwarding destination or when it is larger than the maximum reply message size of the rendezvous port at which the forwarded rendezvous was established, an E_PAR error is returned. cmsgsz may be specified as 0.

If the calling task is not in the termination waiting state for the specified rendezvous, an E_OBJ error is returned as a return value. Besides, the value specified in rdvno can not be interpreted as a rendezvous number.

When calptn is specified as 0, an E_PAR error is returned as a return value.

rpl_rdv Terminate rendezvous

【Format】

ER ercd = rpl_rdv(RDVNO rdvno, VP msg, UINT rmsgsz) ;

【Parameter】

RDVNO	rdvno	Rendezvous number to be terminated
VP	msg	Reply message start address
UINT	rmsgsz	Reply message size (in bytes)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (msg, rmsgsz is invalid)
E_CTX	Context error (invoked from others except task)
E_OBJ	Object state error (rdvno is invalid)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call terminates a rendezvous with the assigned rendezvous number specified by rdvno. The message stored in the memory area starting from the address specified by msg and whose size in bytes is specified by rmsgsz will be the reply message. The others except the task accepting rendezvous also might terminates the rendezvous by this system call

If there is a calling task for the rendezvous specified by rdvno in the termination waiting state for the specified rendezvous, the system call copies the reply message specified by msg and rmsgsz to the area where the calling task stores the reply message, then releases that task from waiting. Besides, the released task receives the reply message size (rmsgsz) as a return value of the system call that placed the task in the WAITING state.

If the calling task for the rendezvous specified by rdvno is not in the termination waiting state for the specified rendezvous, an E_OBJ error is returned as a return value. Moreover, if the value specified in rdvno can not be interpreted as a rendezvous number, an E_OBJ error is also returned as a return value.

When rmsgsz is larger than the maximum reply message size of the rendezvous port at which the forwarded rendezvous was established, an E_PAR is returned as a return value. rmsgsz may be specified as 0.

ref_por Reference rendezvous port state

【Format】

ER ercd = ref_por(ID porid, T_RPOR*pk_rpor) ;

【Parameter】

ID	porid	ID number of the rendezvous port to be referenced
T_RPOR*	pk_rpor	Pointer to the packet returning the rendezvous port state.

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rpor includes (T_RPOR type)

ID	ctskid	ID number of the first task in the rendezvous port's call-wait queue.
ID	atskid	ID number of the first task in the rendezvous port's accept-wait queue.

【Error code】

E_ID	Invalid ID number (porid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified rendezvous port is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the rendezvous port with an ID number specified by porid, then returns the state through the packet specified by pk_rpor.

The ID number of the task at the head of the specified rendezvous port's call-wait queue is returned through ctskid. If no tasks are waiting at the rendezvous calling waiting state, TSK_NONE (=0) is returned.

The ID number of the first task at the head of the specified rendezvous port's accept-wait queue is returned through atskid. If no tasks are waiting at the rendezvous accepting waiting state, TSK_NONE (=0) is returned

ref_rdv Reference rendezvous state

【Format】

ER ercd = ref_rdv(RDVNO rdvno, T_RRDV*pk_rrdv) ;

【Parameter】

RDVNO	rdvno	Rendezvous number of the rendezvous to be referenced
T_RRDV*	pk_rrdv	Pointer to the packet returning the rendezvous state.

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rrdv includes (T_RRDV type)

ID	wtskid	ID number of the task in the rendezvous termination waiting state

【Error code】

E_CTX	Context error (invoked from the interrupt handler)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the rendezvous with the assigned rendezvous number specified by rdvno, then returns the state through the packet specified by pk_rrdv.

If the task that has called the rendezvous specified by rdvno is in the termination waiting state for the specified rendezvous, the ID number of that task is returned through wtskid. If the task is not in the termination waiting state for the specified rendezvous and if the value specififed by rdvno can not be interpreted as a rendezvous number , TSK_NONE (＝0) is returned through wtskid.

5.5 Memory pool management functions

5.5.1 Fixed-length memory pool

cre_mpf	**Create fixed-length memory pool**
acre_mpf	**Create fixed-length memory pool (Automatic ID assignment)**

【Format】

ER ercd = cre_mpf(ID mpfid, T_CMPF*pk_cmpf) ;
ER_ID mpfid = acre_mpf(T_CMPF*pk_cmpf) ;

【Parameter】

ID	mpfid	ID number of the fixed-length memory pool to be created (except acre_mpf)
T_CMPF*	pk_cmpf	Pointer to the packet containing the fixed-length memory pool creation information

pk_cmpf includes (T_CMPF type)

ATR	mpfatr	fixed-length memory pool attribute
UINT	blkcnt	Number of memory blocks that can be acquired (in bytes)
UINT	blksz	Size of the memory block (in bytes)
VP	mpf	fixed-length memory pool start address
VB const *	name	fixed-length memory pool name (character string)

【Return value】

In case of cre_mpf

ER	ercd	Successful completion (E_OK) or error code

In case of acre_mpf

ER_ID	mpfid	ID number (positive value) of the created fixed-length memory pool or error code

【Error code】

E_RSATR	Reserved attribute (mpfatr is invalid or unusable)
E_PAR	Parameter error (blkcnt, blksz is invalid)
E_ID	Invalid ID number (mpfid is invalid or unusable ; only cre_mpf)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (control area or memory pool area can not be allocated)

E_NOID	Lack of ID number (there is no fixed-length memory pool ID assignable ; only acre_mpf)
E_OBJ	Object state error (specified fixed-length memory pool is already registered ; only cre_mpf)

【Invoking context】	cre_mpf	acre_mpf
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

This system call creates the fixed-length memory pool with an ID number specified by mpfid based on the fixed-length memory pool creation information specified by pk_cmpf. mpfatr is the attribute of the fixed-length memory pool, blkcnt is the number of memory blocks that can be acquired form the fixed-length memory pool, blksz is the size (in bytes) of the memory block to acquire, mpf is the start address of the fixed-length memory pool area.

acre_mpf assigns an ID number with biggest value from the pool of unassigned fixed-length memory pool IDs for the fixed-length memory pool to be created and returns the assigned ID number as a return value.

mpfatr can be specified as (TA_TFIFO || TA_TPRI). The fixed-length memory pool's wait queue will be in FIFO order if TA_TFIFO (＝0x00) is specified, it will be in the task priority order if TA_TPRI (＝0x01) is specified.

The necessary area to be able to acquire blkcnt number memory blocks with blksz size starts from the address specified by mpf is used as the memory pool area. An application program can estimate the necessary size be able to acquire blkcnt number of memory blocks with blksz size by using the TSZ_MBF. If mbf is specified as NULL (= 0), it is allocated automatically from the memory area used for defined memory pool at the time of configufation. If blkcnt or blksz is specified as 0, an E_PAR error is returned as a return value.

del_mpf Delete fixed-length memory pool

【Format】

ER ercd = del_mpf(ID mpfid) ;

【Parameter】

ID	mpfid	ID number of the fixed-length memory pool to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mpfid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified fixed-length memory pool is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the fixed-length memory pool with an ID number specified by mpfid.

get_mpf	**Acquire fixed-length memory block**
pget_mpf	**Acquire fixed-length memory block (Polling)**
tget_mpf	**Acquire fixed-length memory block (with Timeout)**

【Format】

ER ercd = get_mpf(ID mpfid, VP *p_blk) ;
ER ercd = pget_mpf(ID mpfid, VP *p_blk) ;
ER ercd = tget_mpf(ID mpfid, VP *p_blk, TMO tmout) ;

【Parameter】

ID	mpfid	ID number of the fixed-length memory pool from which a memory block is acquired
TMO	tmout	Specified timeout (only tget_mpf)

【Return value】

ER	ercd	Successful completion (E_OK) or error code
VP	blk	Start address of the acquired memory block

【Error code】

E_ID	Invalid ID number (mpfid is invalid or unusable)
E_CTX	Context error (invoked from others except task：except pget_mpf, invoked from the interrupt handler：only pget_mpf, dispatch pending state：except pget_mpf)
E_NOEXS	Non-existent object (specified fixed-length memory pool is not registered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting；except pget_mpf)
E_TMOUT	Polling failure or timeout (except get_mpf)
E_DLT	Waiting object deleted (the fixed-length memory pool is deleted while waiting; except pget_mpf0

【Invoking context】	**get_mpf**	**pget_mpf**	**tget_pmf**
Task	Possible	Possible	Possible
Initialization handler	Impossible	Possible	Impossible
Time event handler	Impossible	Possible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

These system call selects one of the memory blocks and takes on acquired status if free memory blocks are available in the memory area of the fixed-length memory pool with an ID

number specified by mpfid, then returns the start address through blk. If there are no free memory blocks, the invoking task will be placed in the wait queue and moved to the acquiring waiting state for the fixed-length memory block.

pget_mpf is the polling system call execute the process of get_mpf, tget_mpf is the system call as get_mpf with an additional timeout feature. Moreover, tmout can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, tget_mpf that set tmout to TMO_FEVR will be used as get_mpf, tget_mpf that set tmout to TMO_POL will be used as pget_mpf.

rel_mpf Release fixed-length memory block

【Format】

ER ercd = rel_mpf(ID mpfid, VP blk) ;

【Parameter】

ID	mpfid	ID number of the fixed-length memory pool to which the memory block is released
VP	blk	Start address of the memory block to be released

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mpfid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified fixed-length memory pool is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call releases the memory block with the start address specified by blk to the memory area of the fixed-length memory pool, when there are no task waiting to acquire the memory block in the fixed-length memory pool with an ID number specified by mpfid.

If there are task waiting for acquiring, the system call lets the first task in the wait queue acquire the released memory block, then release that task from waiting. Besides, the released task receives E_OK as a return value of the system call that placed the task in the WAITING state, and also receives the blk value as the start address of the memory block that acquired from the fixed-length memory block.

The start address of the memory block to be released must be the one which has not been released, because it is the one which returns the start address of the memory block that acquired from the fixed-length memory pool specified by mpfid.

ref_mpf Reference fixed-length memory pool state

【Format】

ER ercd = ref_mpf(ID mpfid, T_RMPF *pk_rmpf) ;

【Parameter】

ID	mpfid	ID number of the fixed-length memory pool to be referenced
T_RMPF*	pk_rmpf	Pointer to the packet returning the fixed-length memory pool state.

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rmpf includes (T_RMPF type)

ID	wtskid	ID number of the first task in the fixed-length memory pool' wait queue
UINT	fblkcnt	Number of free memory blocks of the fixed-length memory pool (quantity)

【Error code】

E_ID	Invalid ID number (mpfid is invalid , or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified fixed-length memory pool is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the fixed-length memory pool with an ID number specified by mpfid, and returns the state through the packet specified by pk_rmpf.

The ID number of the task at the head of the specified fixed-length memory pool 's wait queue is returned through wtskid. If no tasks are waiting to acquire the memory block, TSK_NONE (=0) is returned.

The number of free memory blocks in the specified fixed-length memory pool area is returned through fblkcnt. The definition name of the fixed-length memory pool ID which is created by the configurator is specified to use through mpfid.

5.5.2 Variable length memory pool

cre_mpl	**Create variable length memory pool**
acre_mpl	**Create variable length memory pool (Automatic ID assignment)**

【Format】

ER ercd = cre_mpl(ID mplid, T_CMPL*pk_cmpl) ;

ER_ID mplid = acre_mpl(T_CMPL*pk_cmpl) ;

【Parameter】

ID	mplid	ID number of the variable length memory pool to be created (except acre_mpl)
T_CMPL*	pk_cmpl	Pointer to the packet containing the variable length memory pool creation information

pk_cmpl includes (T_CMPL type)

ATR	mplatr	variable length memory pool attribute
SIZE	mplsz	Size of the variable length memory pool area (in bytes)
VP	mpl	Start address of the variable length memory pool area
VB const *	name	The variable length memory pool name (character string)

【Return value】

In case of cre_mpl

ER	ercd	Successful completion (E_OK) or error code

In case of acre_mpl

ER_ID	mplid	ID number (positive value) of the created variable length memory pool or error code

【Error code】

E_RSATR	Reserved attribute (mplatr is invalid or unusable)
E_PAR	Parameter error (mplsz is invalid)
E_ID	Invalid ID number (mplid is invalid or unusable ; only cre_mpl)
E_CTX	Context error (invoked from the interrupt handler)
E_NOMEM	Insufficient memory (memory pool area can not be allocated)
E_NOID	Lack of ID number (there is no variable length memory pool ID assignable ; only acre_mpl)
E_OBJ	Object state error (specified variable length memory pool is already registered ; only cre_mpl)

【Invoking context】	cre_mpl	cre_mpl
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

These system call creates the variable length memory pool with an ID number specified by mplid based on the variable length memory pool creation information specified by pk_cmpl. mplatr is the attribute of the variable length memory pool, mplsz is the size of the variable length memory pool area (in bytes), mpl is the start address of the variable length memory pool area.

acre_mpl assigns an ID number with biggest value from the pool of unassigned variable length memory pool IDs for the variable length memory pool to be created and returns the assigned ID number as a return value.

mplatr can be specified as (TA_TFIFO || TA_TPRI). The variable length memory pool's wait queue will be in FIFO order if T_TFIFO (＝0x00) is specified, it will be in the task priority order if TA_TPRI (＝0x01) is specified.

The memory area with mplsz byte size starting from the address specified by mpl is used as the memory pool area. Besides, if mpl is specified as NULL (= 0), the memory area with mplsz byte size is automatically allocated from the memory area used for defined memory pool by the configuration.

If mplsz is specified as 0, an E_PAR error is returned as a return value.

del_mpl Delete variable length memory pool

【Format】

ER ercd = del_mpl(ID mplid) ;

【Parameter】

ID	mplid	ID number of the variable length memory pool to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mplid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified variable length memory pool is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the variable length memory pool with an ID number specified by mplid.

get_mpl	**Acquire variable length memory block**
pget_mpl	**Acquire variable length memory block (Polling)**
tget_mpl	**Acquire variable length memory block (with Timeout)**

【Format】

ER ercd = get_mpl(ID mplid, UINT blksz, VP * p_blk) ;
ER ercd = pget_mpl(ID mplid, UINT blksz, VP * p_blk) ;
ER ercd = tget_mpl(ID mplid, UINT blksz, VP * p_blk, TMO tmout) ;

【Parameter】

ID	mplid	ID number oft the variable length memory pool from which a memory block is acquired
UINT	blksz	Size of the memory block to be acquired (in bytes)
TMO	tmout	Specified timeout (only tget_mpl)

【Return value】

ER	ercd	Successful completion (E_OK) or error code
VP	blk	Stat address of the acquired memory block

【Error code】

E_PAR	Parameter error (blksz is invalid)
E_ID	Invalid ID number (mplid is invalid or unusable)
E_CTX	Context error (invoked from others except task：except pget_mpl, Invoked from the interrupt handler ：only pget_mpl, dispatch pending state ：except pget_mpl)
E_NOEXS	Non-existent object (specified variable length memory pool is not registered)
E_RLWAI	Forced release from waiting (rel_wai is accepted while waiting；except pget_mpl)
E_TMOUT	Polling failure or timeout (except get_mpl)
E_DLT	Waiting object deleted (the specified variable length memory pool is deleted while waiting；except pget_mpl)

【Invoking context】	get_mpl	pget_mpl	tget_mpl
Task	Possible	Possible	Possible
Initialization handler	Impossible	Possible	Impossible
Time event handler	Impossible	Possible	Impossible
Interrupt handler	Impossible	Impossible	Impossible

【Description】

These system call acquires the memory block whose size is specified by blksz from the variable length memory pool with an ID number specified by mplid, and then returns the start address through blk.

Specific actions to be performed depend on whether there is a task waiting to acquire a memory block with precedence over the invoking task. If no task are waiting to acquire a memory block at the specified variable length memory pool, and if all of the waiting tasks in the wait queue of task priority order have lower priority than the invoking task, a memory block of size blksz bytes is acquired form the memory pool area. If this condition is not satisfied, and if the free memory area is insufficient for acquiring a memory block, the invoking task is placed in the wait queue, and moved to the acquiring waiting state for the variable length memory block.

When the first task in the wait queue has changed as the result of the task waiting for acquiring of the variable memory block is released from waiting by rel_wai and ter_tsk, or timeout, the system call checks from the new first task, if possible, makes the task acquires a memory block.

pget_mpl is the polling system call execute the process of get_mpl, tget_mpl is the system call as get_mpl with an additional timeout feature. tmout can be set to a positive number indicating a timeout duration or it can be set to TMO_POL (＝0) or TMO_FEVR (＝－1). In μC3/Standard, tget_mpl that set tmout to TMO_FEVR will be used as get_mpl, tget_mpl that set tmout to TMO_POL will be used as pget_mpl.

If blksz is specified as 0, and the value greater than the maximum size of the memory block to be able to acquire from the variable memory pool is specified in blksz, an E_PAR error is returned as a return value.

rel_mpl Release variable length memory block

【Format】

ER ercd = rel_mpl(ID mplid, VP blk) ;

【Parameter】

ID	mplid	ID number of the variable length memory pool to which the memory block is released
VP	blk	Start address of the memory block to be released

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (mplid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified variable length memory pool is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call releases the memory block with the start address specified by blk to the variable length memory pool with an ID number specified by mplid.

If there are tasks waiting to acquire a memory block in the specified variable length memory pool, as a result of releasing the memory block, the system call checks if the task at the head of the wait queue can acquire a memory block of the requested size. If possible, the system call lets the task acquire the memory block, then releases that task from waiting. At this time, the released task receives E_OK as a return value of the system call that placed the task in the WAITING state, and also receives the start address of the acquired memory block as the start address of the memory block that be acquired from the variable length memory block. Moreover, when some tasks waiting to acquire a memory block still remain, the same action must be repeated on the new task at the head of the wait queue.

The specified variable length memory pool to which the memory block is released must be the variable length memory pool with the same ID number from which the memory block was acquired. Besides, the start address of the memory block to be released is one of the service call of get_mpl, pget_mpl, tget_mpl, and must be the one which has not been released,

because it is the one which is returned as the start address of the acquired memory block.

ref_mpl Reference variable length memory pool state

【Format】

ER ercd = ref_mpl(ID mplid, T_RMPL*pk_rmpl) ;

【Parameter】

ID	mplid	ID number of the variable length memory pool to be referenced
T_RMPL*	pk_rmpl	Pointer to the packet returning the variable length memory pool state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rmpl includes (T_RMPL type)

ID	wtskid	ID number of the first task in the variable length memory pool's wait queue
SIZE	fmplsz	Total size of the free space in the variable length memory pool (in bytes)
UINT	fblksz	Maximum memory block size that can be acquire instantly (in bytes)

【Error code】

E_ID	Invalid ID number (mplid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified variable length memory pool is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the variable length memory pool with an ID number specified by mplid, then returns the state through the packet specified by pk_rmpl.

The ID number of the first task in the variable length memory pool's wait queue is returned through wtskid. If there is no task waiting for acquisition of the memory block, TSK_NONE (=0) is returned. The total size (in bytes) of the current free variable length memory pool area is returned through fmplsz. The maximum memory block size (in bytes) that can be acquired instantly from the specified variable length memory pool is returned through fblksz.

5.6 Time management functions

5.6.1 System time management

set_tim	Set system time

【Format】

```
ER ercd = set_tim(SYSTIM    *p_systim) ;
```

【Parameter】

SYSTIM	systim	Time to set as system time

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_CTX	Context error (invoked from the interrupt handler)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call sets the current system time to the time specified by systim. Also, timeout time of the system call which already has been invoking will not be changed by the changing of the system time.

get_tim Reference system time

【Format】

ER ercd = get_tim(SYSTIM *p_systim) ;

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code
SYSTIM	systim	Current system time

【Error code】

E_CTX	Context error (invoked from the interrupt handler)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call calls the current system time, and returns it through system.

isig_tim Supply time tick

【Format】

```
ER ercd = isig_tim( ) ;
```

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_CTX	Context error (invoked from others except interrupt handler)
E_NOMEM	Insufficient memory (lack of SSB)

【Invoking context】

Task	Impossible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Possible

【Description】

This system call adds time tick to the system time.

5.6.2 Cylic handler

cre_cyc	**Create cyclic handler**
acre_cyc	**Create cyclic handler (Automatic ID assignment)**

【Format】

ER ercd = cre_cyc(ID cycid, T_CCYC*pk_ccyc) ;

ER_ID cycid = acre_cyc(T_CCYC*pk_ccyc) ;

【Parameter】

ID	cycid	ID number of the cyclic handler to be create (except acre_cyc)
T_CCYC*	pk_ccyc	Pointer to the packet containing the cyclic handler creation information

pk_ccyc includes (T_CCYC type)

ATR	cycatr	Cyclic handler attribute
VP_INT	exinf	Cyclic handler extended information
FP	cychdr	Cyclic handler activate address
RELTIM	cyctim	Cyclic handler activation cycle
RELTIM	cycphs	Cyclic handler activation phase
VB const *	name	Cyclic handler name (character string)

【Return value】

In case of cre_cyc

ER	ercd	Successful completion (E_OK) or error code

In case of acre_cyc

ER_ID	cycid	ID number (positive value) of the created cyclic handler or error code

【Error code】

E_RSATR	Reserved attribute (cycatr is invalid or unusable)
E_PAR	Parameter error (cyctim is invalid)
E_ID	Invalid ID number (cycid is invalid or unusable ; only cre_cyc)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (control area can not be allocated)
E_NOID	Lack of ID number (there is no cyclic handler ID assignable ; only acre_cyc)
E_OBJ	Object state error (specified cyclic handler is already registered ; only cre_cyc)

【Invoking context】	cre_cyc	acre_cyc
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

These system call creates a cyclic handler with an ID number specified by cycid based on the cyclic handler creation information specified by pk_ccyc. cycatr is the attribute of the cyclic handler, exinf is the extended information passed as a parameter when the cyclic handler is activated, cychdr is activate address of the cyclic handler, cyctim is the activation cycle of the cyclic handler and cycphs is the activation phase of the cyclic handler.

acre_cyc assigns an ID number with biggest value from the pool of unassigned cyclic handler IDs for the cyclic handler to be created and returns the assigned ID number as a return value.

cycatr can be specified as ((TA_HLNG || TA_ASM)| [TA_STA] | [TA_PHS]). If TA_STA (＝0x02) is specified, the cyclic handler is in operation state when it is created. In the other cases, the cyclic handler is in non-operational state when it is created. If TA_PHS (＝0x04) is specified, the phase at the time of the cyclic handler creation is preserved. In case 0 is indicated in cyctim, it will return E_PAR error as Return value.

If cyctim is specified as 0, an E_PAR error is returned as a return value.

del_cyc Delete cyclic handler

【Format】

ER ercd = del_cyc(ID cycid) ;

【Parameter】

ID	cycid	ID number of the cyclic handler to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (cycid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified cyclic handler is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the cyclic handler with an ID number specified by cycid.

sta_cyc Start cyclic handler operation

【Format】

ER ercd = sta_cyc(ID cycid) ;

【Parameter】

ID	cycid	ID number of the cyclic handler whose operation is to be started

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (cycid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call moves the cyclic handler with an ID number specified by cycid to the operational state. Also, the next activation time is the time when the system call is invoked plus the activation cycle of the cyclic handler. At this time, if it has already been in operational state, only the next activation time is updated. If TA_PHS attribute is specified, the cyclic handler is moved from non-operational state to operational state, while if it is already in operational state, no action is required.

stp_cyc Stop cyclic handler operation

【Format】

ER ercd = stp_cyc(ID cycid) ;

【Parameter】

ID	cycid	ID number of the cyclic handler whose operation is to be stopped

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (cycid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified cyclic handler is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call moves the cyclic handler with an ID number specified by cycid to the non-operational state. If the cyclic handler is ready in non-operational state, no action is required.

ref_cyc Reference cyclic hanlder state

【Format】

ER ercd = ref_cyc(ID cycid, T_RCYC *pk_rcyc) ;

【Parameter】

ID	cycid	ID number of the cyclic handler to be referenced
T_RCYC*	pk_rcyc	Pointer to the packet returning the cyclic handler state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rcyc includes (T_RCYC type)

STAT	cycstat	Cyclic handler operational state
RELTIM	lefttim	Time left before the next activation of the cyclic handler

【Error code】

E_ID	Invalid ID number (cycid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state the cyclic handler with an ID number specified by cycid and returns the state through the packet specified by pk_rcyc.

One of the following value is returned through cycstat depending on the operational state of the cyclic handler.

TCYC_STP	0x00	Cyclic handler is in non-operational state
TCYC_STA	0x01	Cyclic handler is in operational state

If the specified cyclic handler is in operational state, the time left before the specified cyclic handler's next activation time is returned through lefttim. However, the value returned through lefttim is the time preserved till the next activation time. Therefore, if the cyclic handler is activated on the next time tick, 0 will be returned through lefttim. When the specified cyclic handler is in non-operational state, an undefined value will be returned through lefttim.

5.6.3 Alarm handler

cre_alm	**Create alarm handler**
acre_alm	**Create alarm handler (Automatic ID assignment)**

【Format】

ER ercd = cre_alm(ID almid, T_CALM*pk_calm) ;
ER_ID almid = acre_alm(T_CALM*pk_calm) ;

【Parameter】

ID	almid	ID number of the alarm handler to be created (except acre_alm)
T_CALM*	pk_calm	Pointer to the packet containing the alarm handler creation information

pk_caim includes (T_CALM type)

ATR	almatr	Alarm handler attribute
VP_INT	exinf	Alarm handler extended information
FP	almhdr	Alarm handler activate address
VB const *	name	Alarm handler name (character string)

【Return value】

In case of cre_alm

ER	ercd	Successful completion (E_OK) or error code

In case of acre_alm

ER_ID	almid	ID number (positive value) of the created alarm handler or error code

【Error code】

E_RSATR	Reserved attribute (almatr is invalid or unusable)
E_ID	Invalid ID number (almid is invalid or unusable ; only cre_alm)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (control area can not be allocated)
E_NOID	Lack of ID number (there is no alarm handler ID assignable ; only acre_alm)
E_OBJ	Object state error (specified alarm handler is already registered ; only cre_alm)

【Invoking context】	cre_alm	cre_alm
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interrupt handler	Impossible	Impossible

【Description】

These system call creates the alarm handler with an ID number specified by almid based on the alarm handler creation information specified by pk_calm. almatr is the attribute of the alarm handler, exinf is the extended information passed as a parameter when the alarm handler is activated, and almhdr is the activate address of the alarm handler.

acre_alm assigns an ID number with biggest value from the pool of unassigned alarm handler IDs for the alarm handler to be created and returns the assigned ID number as a return value.

Right after the alarm handler is created, the activation time of the alarm handler is not set and the alarm handler is in non-operational state.

almatr can be set as (TA_HLNG) only.

del_alm Delete alarm handler

【Format】

ER ercd = del_alm(ID almid) ;

【Parameter】

ID	almid	ID number of the alarm handler to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (almid is invalid or unusable)
E_CTX	Context error (invoked from the others except task)
E_NOEXS	Non-existent object (specified alarm handler is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call deletes the alarm handler with an ID number specified by almid.

sta_alm Start alarm handler operation

【Format】

ER ercd = sta_alm(ID almid, RELTIM almtim) ;

【Parameter】

ID	almid	ID number of the alarm handler whose operation is to be started
RELTIM	almtim	Activation time of the alarm handler (relative time)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (almtim is invalid)
E_ID	Invalid ID number (almid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified alarm handler is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call sets the activation time of the alarm handler with an ID number specified by almid after the relative time specified by almid since the system call is invoked and then starts the alarm's operation

If the alarm handler is already in an operational state, the previous activation time is deleted, and a new one is set.

stp_alm Stop alarm handler operation

【Format】

ER ercd = stp_alm(ID almid) ;

【Parameter】

ID	almid	ID number of the alarm handler whose operation is to be stopped

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (almid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified alarm handler is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call releases the activation time of the alarm handler with an ID number specified by almid and then moves the alarm handler to a non-operational state. In case the alarm handler is already in a non-operational state, no action is required.

ref_alm Reference alarm handler state

【Format】

ER ercd = ref_alm(ID almid, T_RALM*pk_ralm) ;

【Parameter】

ID	almid	ID number of the alarm handler to be referenced
T_RALM*	pk_ralm	Pointer to the packet returning the alarm handler state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_ralm includes (T_RALM type)

STAT	almstat	Alarm handler operational state
RELTIM	lefttim	Amount of time until the activation time of the alarm handler

【Error code】

E_ID	Invalid ID number (almid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_NOEXS	Non-existent object (specified alarm handler is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the alarm handler with an ID number specified by almid and returns the packet through the packet specified by pk_ralm.

One of the following values will be returned through almstat depending on the operational state of the specified alarm handler. .

TALM_STP	0x00	Alarm handler is in non-operational state
TALM_STA	0x01	Alarm handler is in operational state

The relative time until the specified alarm handler's activation time is returned through lefttim if the specified handler is in operational state. However, the value returned through lefttim is the time preserved until the specified alarm handler is activated. Thus, in case the alarm handler is activated on the next time tick, 0 is returned through lefttim. If the specified alarm handler is in non-operational state, the value returned through lefttim is undefined value.

5.6.4 Overrun handler

def_ovr Define overrun handler

【Format】

ER ercd = def_ovr(T_DOVR * pk_dovr) ;

【Parameter】

T_DOVR*	pk_dovr	Pointer to the packet containing the overrun handler definition information

pk_dovr includes (T_DOVR type)

ATR	ovratr	Overrun handler attribute
FP	ovrhdr	Activate address of the overrun handler

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_RSATR	Reserved attribute (ovratr is invalid or unusable)
E_CTX	Context error (invoked from others except task and initialization handler)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call defines the overrun handler based on the overrun handler definition information specified by pk_dovr. ovratr is the attribute of the overrun handler and ovrhdr is the activate address of the overrun handler.

If pk_dovr is specified as NULL (＝0), the definition of the overrun handler currently defined is deleted, and the overrun handler enters into undefined state.

At this time, the processing time limit for all tasks also released, Besides, when a new overrun handler is defined over top of an old one, the old one will released and replaced by the new one. The processing time limit for the task will not be released.

ovratr can be set as (TA_HLNG) only.

ivsig_ovr

【Format】

ER ercd = ivsig_ovr( ) ;

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_CTX	Context error (invoked from others except interrupt handler)
E_NOMEM	Insufficient memory (lack of SSB)

【Invoking context】

Task	Impossible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Possible

【Description】

In case the overrun handler's operation of the running task is activating, the processing time used is updated.

sta_ovr Start overrun handler operation

【Format】

ER ercd = sta_ovr(ID tskid, OVRTIM ovrtim) ;

【Parameter】

ID	tskid	ID number of the task where the overrun handler should start operation
OVRTIM	ovrtim	Processing time limit for the task to be set

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_OBJ	Object state error (overrun handler is not defined)
E_NOEXS	Non-existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call starts the operation of the overrun handler for the task with an ID number specified by tskid. Specifically, it sets the processing time limit for the specified task as the time specified by ovrtim and then clears the processing time used to 0.

In case the task that already has a processing time limit is specified, the old processing time limit will be deleted, and the new processing time limit will be set. At this time, the processing time used will be cleared to 0.

If TSK_SELF（=0）is specified in tskid, the invoking task will be the specified task.

stp_ovr Stop overrun handler operation

【Format】

ER ercd = stp_ovr(ID tskid) ;

【Parameter】

ID	tskid	ID number of the task where the overrun handler should stop operation

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_OBJ	Object state error (overrun handler is not defined)
E_NOEXS	Non-existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call stops the operation of the overrun handler for the task with an ID number specified by tskid. In case the task that does not have a processing time limit is specified, no action is required.

If TSK_SELF (=0) is specified in tskid, the invoking task will be the specified task.

ref_ovr Reference overrun handler state

【Format】

ER ercd = ref_ovr(ID tskid, T_ROVR*pk_rovr) ;

【Parameter】

ID	tskid	ID number of the task for which the overrun handler's state should be referenced
T_ROVR*	pk_rovr	Pointer to the packet returning the overrun handler state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rovr includes (T_ROVR type)

STAT	ovrstat	Overrun handler operation state
OVRTIM	leftotm	Remaining processing time

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)
E_CTX	Context error (invoked from the interrupt handler)
E_OBJ	Object state error (overrun handler is not defined)
E_NOEXS	Non-existent object (specified task is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the state of the overrun handler for the task with an ID number specified by tskid and returns the state through the packet specified by pk_rovr .

the operation state of the overrun handler for the specified task is returned through ovrstat.

TOVR_STP	0x00	Processing time limit is not set
TOVR_STA	0x01	Processing time limit is set

If the processing time limit is set for the specified task, the processing time remaining until the overrun handler is activated caused by the specified task is returned through leftotm. Specifically, the value returned is the processing time limit of the specified task minus the processing time used. Besides, if the processing time limit for the specified task is not specified, the undefined value is returned.

If tskid is specified as TSK_SELF (=0), the invoking task will be the specified task.

5.7 System state management functions

rot_rdq	Rotate task precedence
irot_rdq	

【Format】

ER ercd = rot_rdq(PRI tskpri) ;

ER ercd = irot_rdq(PRI tskpri) ;

【Parameter】

PRI	tskpri	Priority of the task whose precedence is rotated

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (tskpri is invalid)
E_NOMEM	Insufficient memory (lack of SSB ; invoked from the interrupt handler)

【Invoking context】	rot_rdq	irot_rdq
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

These system call rotates the precedence of the tasks with the priority specified by tskpri. In other words, the task with the highest precedence of all the runnable tasks with the specified priority will have the lowest precedence among task with the same priority. If TPRI_SELF (=0) is specified in tskpri, the base priority of the invoking task will become the target priority.

【Recommendation】

rot_rdq and irot_rdq in μC3/Standard are the same system call, and both can be used by the same method irregardless of the invoking context. However, it is recommended that rot_rdq is used when invoking from the task context and irot_rdq should be used for the remains.

get_tid	Reference task ID in the RUNNING state
iget_tid	

【Format】

ER ercd = get_tid(ID *p_tskid) ;
ER ercd = iget_tid(ID *p_tskid) ;

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code
ID	tskid	ID number of the task in the running state

【Error code】

No specific errors

【Invoking context】	get_tid	iget_tid
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

These system call references the ID number of the task in the RUNNING state, and returns the task ID through tskid. If the system call is invoked from non-task contexts, TSK_NONE (= 0) is returned through tskid when there is no task in the RUNNING state. .

【Recommendation】

get_tid and iget_tid in μC3/Standard are the same system call, and both can be used by the same method irregardless of the invoking context. However, it is recommended that get_tid is used when invoking from the task context and iget_tid should be used for the remains.

loc_cpu	Transit to CPU locked state
iloc_cpu	

【Format】

ER ercd = loc_cpu( ) ;

ER ercd = iloc_cpu( ) ;

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

No specific errors

【Invoking context】	loc_cpu	iloc_cpu
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

These system call transits the system to the CPU locked state. If the system is in the CPU locked state, no action is required. .

The CPU locked state depends on the processor, so please refer to “The manual of processor-dependent parts” for more details.

【Recommendation】

loc_cpu and iloc_cpu in µC3/Standard are the same system call, and both can be used by the same method irregardless of the invoking context. However, it is recommended that loc_cpu is used when invoking from the task context and iloc_cpu should be used for the remains.

unl_cpu	CPU unlocked state
iunl_cpu	

【Format】

ER ercd = unl_cpu( ) ;

ER ercd = iunl_cpu( ) ;

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

No specific errors

【Invoking context】	unl_cpu	iunl_cpu
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Possible	Possible
Interrupt handler	Possible	Possible

【Description】

These system call transits the system to the CPU unlocked state. If the system is in the CPU unlocked state, no action is required. .

The CPU unlocked state depends on the processor, so please refer to “The manual of processor-dependent parts”for more details.

【Recommendation】

unl_cpu and iunl_cpu in μC3/Standard are the same system call, and both can be used by the same method irregardless of the invoking context. However, it is recommended that unl_cpu is used when invoking from the task context and iunl_cpu should be used for the remains.

dis_dsp Disable dispatching

【Format】

ER ercd = dis_dsp( ) ;

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_CTX	Context error (invoked from others except task)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call transits the system to the dispatching disabled state. No action required if invoked on the dispatching disabled state.

ena_dsp Enable dispatching

【Format】

ER ercd = ena_dsp( ) ;

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_CTX	Context error (invoked from others except task)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interrupt handler	Impossible

【Description】

This system call transits the system to the dispatching enabled state. No action required if invoked on dispatching enabled state,.

sns_ctx	Reference contexts

【Format】

BOOL state = sns_ctx( ) ;

【Parameter】

None

【Return value】

BOOL	state	Context

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Possible

【Description】

This system call returns TRUE if invoked from non-task contexts and returns FALSE if invoked from task contexts.

sns_loc Reference CPU locked state

【Format】

```
BOOL state = sns_loc( ) ;
```

【Parameter】

None

【Return value】

BOOL	state	CPU locked state

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Possible

【Description】

This system call returns TRUE if the system is in the CPU locked state and returns FALSE if the system is in the CPU unlocked state.

sns_dsp Reference dispatching disabled state

【Format】

```
BOOL state = sns_dsp( ) ;
```

【Parameter】

None

【Return value】

BOOL	state	Dispatching disabled state

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Possible

【Description】

This system call returns TRUE if the system is in the dispatching disabled state and returns FALSE if the system is in the dispatching enabled state.

sns_dpn Reference dispatch pending state

【Format】

```
BOOL state = sns_dpn( ) ;
```

【Parameter】

None

【Return value】

BOOL	state	Disapatch pending state

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Possible

【Description】

This system call returns TRUE if the system is in the dispatch pending state and returns FALSE if the system is in any other state. In other words, it returns TRUE if the system is in either CPU locked state, or dispatching disabled state, or when interrupt level is higher than task level.

ref_sys Reference system state

【Format】

ER ercd = ref_sys(T_RSYS *pk_rsys) ;

【Parameter】

T_RSYS*	pk_rsys	Pointer to the packet returning the system state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_rsys includes (T_RSYS type)

SIZE	fsyssz	Total size of the system memory free space
SIZE	fstksz	Total size of the memory free space for stack
SIZE	fmplsz	Total size of the memory free space for memory pool
UH	utskid	Number of task ID which is already created
UH	usemid	Number of semaphore ID which is already created
UH	uflgid	Number of eventflag ID which is already created
UH	udtqid	Number of data queue ID which is already created
UH	umbxid	Number of mailbox ID which is already created
UH	umbfid	Number of message buffer ID which is already created
UH	uporid	Number of rendezvous port ID which is already created
UH	umpfid	Number of fixed-length memory pool ID which is already created
UH	umplid	Number of variable length memory pool ID which is already created
UH	ualmid	Number of alarm handler ID which is already created
UH	ucycid	Number of cyclic handler ID which is already created
UH	uisrid	Number of interrupt service routine ID which is already created
UH	ssbcnt	Number of minimum SSB

【Error code】

E_CTX	Context error (invoked from the interrupt handler)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call references the system state, and returns the state through the packed specified by pk_rsys.

5.8 Interrupt management functions

def_inh Define interrupt handler

【Format】

ER ercd = def_inh(INHNO inhno, T_DINH*pk_dinh) ;

【Parameter】

INHNO	inhno	Interrupt handler number to be defined
T_DINH*	pk_dinh	Pointer to the packet containing the interrupt handler creation information

pk_dinh includes (T_DINHtype)

ATR	inhatr	Interrupt handler attribute
FP	inthdr	Interrupt handler activate address
IMASK	imask	Interrupt level of the interrupt handler

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (inhno, imask is invalid)
E_NOMEM	Insufficient memory (control area can not be allocated)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interrupt handler	Impossible

【Description】

This system call defines an interrupt handler to the interrupt handler number specified by inhno based on the interrupt handler definition information specified by pk_dinh. inhatr is the attribute of the interrupt handler, inthdr is the interrupt handler activate address, and imask is the interrupt level of the interrupt handler. This interrupt handler number depends on the processor, so please refer to “The manual of processor-dependent parts”for more details.

If pk_dinh is specified as NULL (＝0), the interrupt handler currently defined is released. Besides, when a new interrupt handler is defined over top of an old one, the old one is replaced by the new one. In μC3/Standard, the interrupt number of the interrupt service routine and the interrupt handler number have the same meaning. Therefore, if the interrupt handler is defined to the interrupt number which is already used for the interrupt service routine, an E_PAR error is returned as a return value.

cre_isr	Create interrupt service routine
acre_isr	Create interrupt service routine (Automatic ID assignment)

【Format】

ER ercd = cre_isr(ID isrid, T_CISR*pk_cisr) ;

ER_ID isrid = acre_isr(T_CISR*pk_cisr) ;

【Parameter】

ID	isrid	ID number of the interrupt service routine to be create (only cre_isr)
T_CISR*	pk_cisr	Pointer to the packet containing the interrupt service routine creation information

pk_cisr includes (T_CISR type)

ATR	isratr	Interrupt service routine attribute
VP_INT	exinf	Interrupt service routine extended information
INTNO	intno	Interrupt number to which the interrupt service routine to be attached
FP	isr	Interrupt service routine activate address
IMASK	imask	Interrupt level of the interrupt service routine

【Return value】

In case of cre_isr

ER	ercd	Successful completion (E_OK) or error code

In case of acre_isr

ER_ID	isrid	ID number (positive value) of the created interrupt service routine or error code

【Error code】

E_PAR	Parameter error (intno, imask is invalid)
E_ID	Invalid ID number (isrid is invalid or unusable ; only cre_isr)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (control area cannot be allocated)
E_NOID	Lack of ID number (there is no interrupt service handler ID assignable; only acre_isr)
E_OBJ	Object state error (specified interrupt service routine handler is already registered; only cre_isr)

【Invoking context】	cre_isr	acre_isr
Task	Possible	Possible
Initialization handler	Possible	Possible
Time event handler	Impossible	Impossible
Interruption handler	Impossible	Impossible

【Description】

These system calls creates the interrupt service routine with an ID number specified by isrid based on the interrupt service routine creation information specified by pk_cisr. isratr is the attribute the interrupt service routine, exinf is the extended information passed as a parameter when interrupt service routine is activated, intno is the interrupt number specified for the interrupt which starts the interrupt service routine, isr is the activate address of the interrupt service routine and imask is the interrupt level of the interrupt handler.

This interrupt number depends on the processor. For more details, please make reference to “The manual of processor-dependent parts”.

acre_isr assigns an ID number with biggest value from the pool of unassigned interrupt service routine IDs for the interrupt service routine to be created and returns the assigned ID number as a return value.

In μC3/Standard, the interrupt number of the interrupt service routine and the interrupt handler number have the same meaning. Therefore, in case of trying to create the interrupt service routine with the interrupt handler number which is already used for the interrupt handler, an E_PAR error is returned as a return value

isratr can be specified as (TA_HLNG) only.

del_isr Delete interrupt service routine

【Format】

ER ercd = del_isr(ID isrid) ;

【Parameter】

ID	isrid	ID number of the interrupt service routine to be deleted

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (isrid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified interrupt service routine is not registered)

【Invoking context】

Task	Possible
Initialization handler	Impossible
Time event handler	Impossible
Interruption handler	Impossible

【Description】

This system call deletes the interrupt service routine with an ID number specified by isrid.

ref_isr Reference interrupt service routine state

【Format】

ER ercd = ref_isr(ID isrid, T_RISR*pk_risr) ;

【Parameter】

ID	isrid	ID number of the interrupt service routine to be referenced
T_RISR*	pk_risr	Pointer to the packet returning the interrupt service routine state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_risr includes (T_RISR type)

INTNO	intno	Interrupt number to which the interrupt service routine to be attached
FP	isr	The activate address of the interrupt service routine

【Error code】

E_ID	Invalid ID number (isrid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (specified interrupt service routine is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interruption handler	Impossible

【Description】

This system call references the state of the interrupt service routine with an ID number specified by isrid and then returns the state through the packet specified by pk_risr.

This interrupt number depends on the processor. For more details, please make reference to “The manual of processor-dependent parts”.

dis_int Disable interrupt

【Format】

ER ercd = dis_int(INTNO intno) ;

【Parameter】

INTNO	intno	Interrupt number to be disabled

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (intno is invalid)

【Invoking context】

Task	Impossible
Initialization handler	Impossible
Time event handler	Impossible
Interruption handler	Impossible

【Description】

This system call depends on the processor, including its implementation. For more details, please make reference to “The manual of processor-dependent parts”.

ena_int Enable interrupt

【Format】

ER ercd = ena_int(INTNO intno) ;

【Parameter】

INTNO	intno	Interrupt number to be enabled

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (intno is invalid)

【Invoking context】

Task	Impossible
Initialization handler	Impossible
Time event handler	Impossible
Interruption handler	Impossible

【Description】

This system call depends on the processor, including its implementation. For more details, please make reference to “The manual of processor-dependent parts”.

chg_ims　　Change interrupt mask

【Format】

ER ercd = chg_ims(IMASK imask) ;

【Parameter】

IMASK	imask	Interrupt handler after being changed

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

No specific errors

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interruption handler	Possible

【Description】

This system call changes the processor's interrupt mask level to the value specified by imask. This interrupt mask level depends on the processor, please make reference to "The manual of processor-dependent parts" for more details.

get_ims Reference interrupt mask

【Format】

ER ercd = get_ims(IMASK *p_imask) ;

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code
IMASK	imask	Current interrupt mask

【Error code】

No specific errors

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interruption handler	Possible

【Description】

This system call references the processor's interrupt mask level, and returns it through imask. This interrupt mask level depends on the processor, please make reference to "The manual of processor-dependent parts" for more details.

5.9 System configuration management functions

def_exc Define CPU exception handler

【Format】

ER ercd = def_exc(EXCNO excno, T_DEXC*pk_dexc) ;

【Parameter】

EXCNO	excno	CPU exception handler number to be defined
T_DEXC*	pk_dexc	Pointer to the packet containing the CPU exception handler definition information

pk_dexc includes (T_DEXC type)

ATR	excatr	CPU exception handler attribute
FP	exchdr	CPU exception handler activate address

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error (excno is invalid)
E_CTX	Context error (invoked from others except task)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interruption handler	Impossible

【Description】

This system call defines the CPU exception handler to the CPU exception handler number specified by excno based on the CPU exception handler definition information specified by pk_dexc. excatr is the attribute of the CPU exception handler, exchdr is the activate address of the CPU exception handler.

If NULL (=0) is specified in pk_dexc, the definition of the CPU exception handler currently defined is deleted.. Besides, when a new CPU exception handler is defined over top of an old one, the old one is replaced by the new one.

This system call depends on the processor, including its implementation, please make reference to “The manual of processor-dependent parts” for more details.

ref_cfg Reference configuration information

【Format】

ER ercd = ref_cfg(T_RCFG * pk_rcfg) ;

【Parameter】

T_RCFG*	pk_rcfg	Pointer to the packet returning the configuration information

【Return value】

ER	ercd	Successful completion (E_OK) or error code
pk_rcfg includes (T_RCFG type)		
UH	tskpri_max	Maximum value of task priority
UH	tskid_max	Maximum value of task ID
UH	semid_max	Maximum value of semaphore ID
UH	flgid_max	Maximum value of event flag ID
UH	dtqid_max	Maximum value of data queue ID
UH	mbxid_max	Maximum value of mailbox ID
UH	mbfid_max	Maximum value of message buffer ID
UH	porid_max	Maximum value of rendezvous port ID
UH	mpfid_max	Maximum value of fixed-length memory pool ID
UH	mplid_max	Maximum value of variable-length memory pool ID
UH	almid_max	Maximum value of alarm handler ID
UH	cycid_max	Maximum value of cyclic handler ID
UH	isrid_max	Maximum value of interrupt service routine ID
UH	devid_max	Maximum value of device driver ID
UH	tick	Cycle time of time tick
UH	ssb_cnt	Number of SSB to be created

【Error code】

No specific errors

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interruption handler	Possible

【Description】

This system call references the system configuration, and returns it through the packet specified by pk_rcfg.

ref_ver Reference version information

【Format】

ER ercd = ref_ver(T_RVER *pk_rver) ;

【Parameter】

T_RVER *	pk_rver	Pointer to the packet returning the version information

【Return value】

ER	ercd	Successful completion (E_OK) or error code
pk_rver includes (T_RVER type)		
UH	maker	Maker code of the kernel
UH	prid	Identification number of the kernel
UH	spver	Version number of the ITRON specification
UH	prver	Version number of the kernel
UH	prno[4]	Management information of the kernel product

【Error code】

No specific errors

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interruption handler	Possible

【Description】

This system call references the version information of the using kernel, and returns it through the packet specified by pk_rver.

【Supplement】

At the time of writing this Manual, the maker code has not been acquired yet. Therefore, 0x000 will be returned as the maker code.

5.10 Peculiar functions

5.10.1 Device driver management function

vdef_dev	Define device driver

【Format】

ER ercd = vdef_dev(ID devid, T_CDEV * pk_cdev) ;

【Parameter】

ID	devid	ID number of the device driver to be defined
T_CDEV*	pk_ cdev	Pointer to the packet containing the device driver creation information

pk_cdev includes (T_CDEV type)

VP	ctrblk	Start address of the control information packet
FP	devhdr	Device driver activate address
VB const *	name	Device driver name (character string)

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (devid is invalid or unusable)
E_CTX	Context error (invoked from others except task and initialization handler)
E_NOMEM	Insufficient memory (control area cannot be allocated)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Impossible
Interruption handler	Impossible

【Description】

This system call creates the device driver with an ID number specified by devid based on the device driver creation information specified by pk_cdev. ctrblk is the control information of the device driver, devhdr is the activate address of the device driver.

This system call is implemented for the standard COM driver, it is not the function that force the user to use.

vctr_dev Control device driver

【Format】

ER ercd = vctr_dev(ID devid, ID funcid, VP ctrdev) ;

【Parameter】

ID	devid	Device driver ID number
ID	funcid	Device driver function code
VP	ctrdev	Start address of the device control information packet

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (devid is invalid or unusable)
E_CTX	Context error (invoked from others except task)
E_NOEXS	Non-existent object (device driver is not registered)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Impossible
Interruption handler	Impossible

【Description】

This system call starts the device driver with an ID number specified by devid. funcid is the function code of the device driver, ctrdev the device control information packet.

The device driver will receives the function code through the 1st argument, the start address of the device control information packet through 2nd argument and the start address of the control information packet through 3rd argument.

This system call is implemented for the standard COM driver, it is not the function that force the user to use.

5.10.2 Error handler

vdef_err	Define error handler

【Format】

ER ercd = vdef_err (ATR atr, FP errhdr) ;

【Parameter】

ATR	erratr	Error handler attribute
FP	errhdr	Error handler activate address

【Return value】

ER	ercd	Successful completion (E_OK)

【Invoking context】

Task	Possible
Initialization handler	Possible
Time event handler	Possible
Interruption handler	Possible

【Description】

This system call defines the error handler of the error handler activate address specified by errhdr with the error handler attribute specified by erratr. If the error handler activate address errhdr is specified as NULL (＝0), the definition of the error handler currently defined is deleted.

erratr can be specified as (TA_HLNG) only.

Chapter 6 Explanation of standard COM port driver

6.1 Overview of the standard COM port driver

In μC3/Standard, the usage method of using COM port is defined, and that driver is called the standard COM port driver. Here, the service call of the standard COM port driver is explained.

This system call invoking is available only from task context, and it is impossible to use on the dispatch pending state.

6.2 The service call of the standard COM port driver

ini_com Initialize COM port

【Format】

ER ercd = ini_com(ID DevID, T_COM_SMOD const * pk_SerialMode) ;

【Parameter】

ID	DevID	Device ID number
T_COM_SMOD const *	pk_SerialMode	Pointer to the initialization information packet

pk_SerialMode includes (T_COM_SMOD type)

UW	baud	Baud rate
UB	blen	Data bit
UB	par	Parity
UB	sbit	Stop bit
UB	flow	Flow control

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error
E_ID	Invalid ID number (DevID is invalid or unusable)

【Description】

This service call initializes the device with an ID number specified by DecID according the content of the initialization information packet.

baud is specified as the baud rate of the serial device.

One of the following data bit is specified in blen:

BLEN8	8 bit data length
BLEN7	7 bit data length
BLEN6	6 bit data length
BLEN5	5 bit data length

One of the following parity bit is specified in par.

PAR_NONE	Non-parity bit
PAR_EVEN	Enable even parity bit
PAR_ODD	Enable odd parity bit

One of the following stop bit is specified in sbit.

SBIT1	1 bit stop
SBIT15	1.5 bit data length
SBIT2	2 bit data length

One of the following flow control is specified in flow.

FLW_NONE	Disable flow control
FLW_XON	Enable software flow control
FLW_HARD	Enable hardware flow control

ctr_com Control COM port

【Format】

ER ercd = ctr_com (ID DevID, UH command, TMO tmout) ;

【Parameter】

ID	DevID	Device ID number
UH	command	Control command
TMO	tmout	Specified timeout

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Parameter error
E_ID	Invalid ID number (DevID is invalid or unusable)
E_TMOUT	Polling failure or timeout

【Description】

The service call controls the device with an ID number specified by DecID according the content specified by command.

Command includes the following types and it is possible specify various command by logical disjunction (OR). In this case, the order will be from the upper command.

RST_COM	0xF800	Reset COM port
CLN_TXBUF	0x8000	Waiting for transmission of send buffer
RST_BUF	0x6000	Clear send/receive buffer
RST_TXBUF	0x4000	Clear send buffer
RST_RXBUF	0x2000	Clear receive buffer
STP_COM	0x1800	Sending/receiving prohibition
STP_TX	0x1000	Sending prohibition
STP_RX	0x0800	Receiving prohibition
SND_BRK	0x0400	Breaking character transmission
STA_COM	0x0300	Enable sending/receiving
STA_TX	0x0200	Enable sending
STA_RX	0x0100	Enable receiving
LOC_TX	0x0080	Lock sending
LOC_RX	0x0040	Lock receiving
UNL_TX	0x0020	Unlock sending
UNL_RX	0x0010	Unlock receiving

In case of CLN_TXBUF, tmout is specified to timeout time, and in case of SND_BRK, it is specified to transmission time. It will be ignored in the other case.

putc_com — Send character to COM port

【Format】

ER ercd = putc_com (ID DevID, VB chr, TMO tmout) ;

【Parameter】

ID	DevID	Device ID number
VB	chr	Sending character
TMO	tmout	Specified timeout

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (DevID is invalid or unusable)
E_TMOUT	Polling failure or timeout

【Description】

The service call sends the sending character chr from the device with an ID number specified by DecID.

tmout is specified to the timeout time remaining until the sending termination.

puts_com	Send character string to COM port

【Format】

ER ercd = puts_com (ID DevID, VB const *p_schr, UINT *p_scnt, TMO tmout) ;

【Parameter】

ID	DevID	Device ID number
VB const *	schr	Sending character string
UINT *	scnt	Number of sending character
TMO	tmout	Specified timeout

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (DevID is invalid or unusable)
E_TMOUT	Polling failure or timeout

【Description】

The service call sends the sending character string schr with the sending character count scnt only from the device with an ID number specified by DecID.

tmout is specified to the timeout time remaining until the sending termination.

getc_com — Receive one character from COM port

【Format】

ER ercd = getc_com(ID DevID, VB *p_rbuf, UB *p_sbuf, TMO tmout) ;

【Parameter】

ID	DevID	Device ID number
VB *	rbuf	Receiving character
UB *	sbuf	Receiving status
TMO	tmout	Specified timeout

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (DevID is invalid or unusable)
E_TMOUT	Polling failure or timeout

【Description】

The service call returns the received character through rbuf and returns the receiving status through sbuf from the device with an ID number specified by DecID. At this time, if the receiving status is unnecessary, p_sbuf is specified as 0.

tmout is specified to the timeout time remaining until the receiving termination.

gets_com — Receive character string from COM port

【Format】

```
ER ercd = gets_com(ID DevID, VB *p_rbuf, UB *p_sbuf, INT eos, UINT *p_rcnt,
                   TMO tmout) ;
```

【Parameter】

ID	DevID	Device ID number
VB *	rbuf	Array of receiving character
UB *	sbuf	Array of receiving status
INT	eos	Ending character
UINT	rcnt	Number of receiving character
TMO	tmout	Specified timeout

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Invalid ID number (DevID is invalid or unusable)
E_TMOUT	Polling failure or timeout

【Description】

The service call returns the received character through rbuf and returns the receiving status through sbuf from the device with an ID number specified by DecID. Size of the area that store the receiving data is specified in rcnt, the received character count is returns through rcnt. At this time, if the receiving status is unnecessary, p_sbuf is specified as 0.

The receiving will completed normally if the store area is full, or if the ending character is received, or if the error occurs when receiving status is available.

tmout is specified to the timeout time remaining until the receiving termination.

ref_com Reference COM port state

【Format】

ER ercd = ref_com(ID tskid, T_COM_REF *pk_SerialRef) ;

【Parameter】

ID	DevID	Device ID number
T_COM_REF *	pk_SerialRef	COM port state

【Return value】

ER	ercd	Successful completion (E_OK) or error code

pk_SerialRef includes (T_COM_REF type)

UH	rxcnt	Number of character is already received
UH	txcnti	Number of character is not sent
UH	status	Status

【Error code】

E_ID	Invalid ID number (tskid is invalid or unusable)

【Description】

The service call references the state of the device with an ID number specified by DecID, and returns the state through the packet specified by pk_SerialRef.

The character count that is already received in driver is returned through rxcnt, the character count that is not sent is returned through txcnt.

The following state is returned through status by the value of logical disjunction (OR) depends on the status.

T_COM_EROVB	0x0001	FIFO overrun
T_COM_EROR	0x0002	Overrun error
T_COM_ERP	0x0004	Parity error
T_COM_ERF	0x0008	Flaming error
T_COM_BRK	0x0010	Breaking character receiving
T_COM_TXOFF	0x0020	Send XOFF receiving
T_COM_RXOFF	0x0040	Receive XOFF sending
T_COM_RTS	0x0080	RTS signal active
T_COM_CTS	0x0100	CTS signal active
T_COM_DTR	0x0200	DTR signal active
T_COM_DSR	0x0400	DSR signal active
T_COM_CD	0x0800	CD signal active
T_COM_RI	0x1000	RI signal active
T_COM_ENARX	0x2000	Receive enabled state
T_COM_ENATX	0x4000	Send enabled state
T_COM_INIT	0x8000	COM port is already initialized

Chapter 7 Appendix

7.1 Data types

The data types defined in the μITRON4.0 specification are as follows. (except data types for packets)

B	Signed 8-bit integer
H	Signed 16-bit integer
W	Signed 32-bit integer
UB	Unsigned 8-bit integer
UH	Unsigned 16-bit integer
UW	Unsigned 32-bit integer
VB	8-bit value with unknown data type
VH	16-bit value with unknown data type
VW	32-bit value with unknown data type
VP	Pointer to an unknown data type
FP	Program activate address (pointer)
INT	Signed integer whose natural size is suitable for the processor
UINT	unsigned integer whose natural size is suitable for the processor
BOOL	Boolean value (TRUE or FALSE)
FN	Function code (signed integer)
ER	Error code (signed integer)
ID	Object ID number (signed integer)
ATR	Object attribute (unsigned integer)
STAT	Object state (unsigned integer)
MODE	Service call operational mode (unsigned integer)
PRI	Priority (signed integer)
SIZE	Memory area size (unsigned integer)
TMO	Specified timeout (signed integer, unit of time is 1ms)
RELTIM	Relative time (unsigned integer, unit of time is 1ms)

SYSTEM	System time (unsigned integer, unit of time is 1ms)
VP_INT	Pointer to an unknown data type, or signed integer whose natural size is suitable for the processor
ER_BOOL	Error code or boolean value
ER_ID	Error code or ID number (negative ID number can not presented)
ER_UINT	Error code or an unsigned integer (the number of effective bits for an unsigned integer is one bit shorter than UINT)
TEXPTN	Bit pattern of the task exception reason (unsigned integer)
FLGPTN	Bit pattern of the eventflag (unsigned integer)
T_MSG	Message header for a mailbox
T_MSG_PRI	Message header with a priority for a mailbox
RDVPTN	Bit pattern of the rendezvous condition (unsigned integer)
RDVNO	Rendezvous number
OVRTIM	Processing time (unsigned integer, unit of time is user-defined)
INHNO	Interrupt handler number
INTNO	Interrupt number
IMASK	Interrupt mask
EXCNO	CPU exception handler number

【Supplement】

INT, UINT, VP_INT, TEXPTN, FLGPTN depends on the processor, so please refer to “The manual of processor-dependent parts”for more details.

7.2 Packet formats

（1）Task management functions

Packet format of the task creation information

```
typedef struct t_ctsk {
        ATR         tskatr ;    /* task attribute   */
        VP_INT      exinf ;     /* task extended information */
        FP          task ;      /* task activate address */
        PRI         itskpri ;   /* task initial priority */
        SIZE        stksz ;     /* task stack size (in bytes)*/
        VP          stk ;       /* start address of task stack area */
        VB const *  name ;      /* task name */
} T_CTSK ;
```

Packet format of task state

```
typedef struct t_rtsk {
        STAT        tskstat ;   /* task state*/
        PRI         tskpri ;    /* task current priority */
        PRI         tskbpri ;   /* task base priority */
        STAT        tskwait ;   /* waiting reason*/
        ID          wobjid ;    /* object ID number for which the task is
                                   waiting */
        TMO         lefttmo ;   /* time remaining till timeout */
        UINT        actcnt ;    /* activation request queuing count */
        UINT        wupcnt ;    /* wakeup request queuing count */
        UINT        suscnt ;    /* suspension request nesting count */
} T_RTSK ;
```

Packet format of task state (simplified version)

```
typedef struct t_rtst {
        STAT        tskstat ;   /* task state*/
        STAT        tskwait ;   /* waiting reason */
} T_RTST ;
```

(2) Task exception handling functions

Packet format of task exception handling routine definition inforamation

```
typedef struct t_dtex {
        ATR         texatr ;    /* task exception handling routine attribute */
        FP          texrtn ;    /* task exception handling routine activate
                                   address */
} T_DTEX ;
```

Packet format of task exception handling state

```
typedef struct t_rtex {
        STAT        texstat ;   /* task exception handling state */
        TEXPTN      pndptn ;    /* pending exception reason */
} T_RTEX ;
```

(3) Synchronization and communication functions

Packet format of semaphore creation information

```
typedef struct t_csem {
        ATR         sematr ;    /* semaphore attribute*/
        UINT        isemcnt ;   /* initial semaphore resource count */
        UINT        maxsem ;    /* maximum semaphore resource count */
        VB const *  name ;      /* semaphore name */
} T_CSEM ;
```

Packet format of semaphore state

```
typedef struct t_rsem {
        ID          wtskid ;    /* ID number of the first task in the
                                   semaphore's wait queue*/
        UINT        semcnt ;    /* current semaphore resource count*/
} T_RSEM ;
```

Packet format of eventflag creation information

```
typedef struct t_cflg {
        ATR         flgatr ;    /* eventflag attribute */
        FLGPTN      iflgptn ;   /*initial value of the eventflag bit pattern*/
        VB const *  name ;      /* eventflag name */
} T_CFLG ;
```

Packet format of eventflag state

```
typedef struct t_rflg {
        ID          wtskid ;    /* ID number of the first task in the eventlflag's
                                   wait queue*/
        FLGPTN      flgptn ;    /* current bit pattern of the eventflag*/
} T_RFLG ;
```

Packet format of data queue creation information

```
typedef struct t_cdtq {
        ATR         dtqatr ;    /* data queue attribute */
        UINT        dtqcnt ;    /* capacity of the data queue area (number of
                                   data)*/
        VP          dtq ;       /*start address of the data queue area */
        VB const *  name ;      /* data queue name */
} T_CDTQ ;
```

Packet format of data queue state

```
typedef struct t_rdtq {
        ID          stskid ;    /* ID number of the first task in the data queue's
                                   send-wait queue */
        ID          rtskid ;    /* ID number of the first task in the data queue's
                                   receive-wait queue*/
        UINT        sdtqcnt ;   /* number of data in the data queue*/
} T_RDTQ ;
```

Packet format of mailbox creation information

```
typedef struct t_cmbx {
        ATR         mbxatr ;    /* mailbox attribute */
        PRI         maxmpri ;   /* maximum priority of the message to be sent */
        VP          mprihd ;    /* start address of the area for message queue
                                   header for each priority*/
        VB const *  name ;      /* mailbox name */
} T_CMBX ;
```

Packet format of mailbox state

```
typedef struct t_rmbx {
        ID          wtskid ;    /* ID number of the first task in the wait queue */
        T_MSG*      pk_msg ;    /* start address of the first message packetinthe
                                   message queue*/
} T_RMBX ;
```

（4）Extended Synchronization and communication functions

Packet format of mutex creation information

```
typedef struct t_cmtx {
        ATR         mtxatr ;    /* mutex attribute */
        PRI         ceilpri ;   /* mutex ceiling priority */
        VB const *  name ;      /* mutex name */
} T_CMTX ;
```

Packet format of mutex state

```
typedef struct t_rmtx {
        ID          htskid ;    /* ID number of the task that locks the mutex */
        ID          wtskid ;    /* ID number of the first task in the mutex's wait
                                   queue */
} T_RMTX ;
```

Packet format of message buffer creation information

```
typedef struct t_cmbf {
        ATR         mbfatr ;    /* message buffer attribute */
        UINT        maxmsz ;    /* maximum message size (in bytes) */
        SIZE        mbfsz ;     /* size of the message buffer area (in bytes) */
        VP          mbf ;       /* start address of the message buffer area */
        VB const *  name ;      /* message buffer name */
} T_CMBF ;
```

Packet format of message buffer state

```
typedef struct t_rmbf {
        ID          stskid ;    /* ID number of the first task in the message
                                   buffer's send-wait queue*/
        ID          rtskid ;    /* ID number of the first task in the message
                                   buffer's receive -wait queue*/
        UINT        smsgcnt ;   /* number of message in the message buffer*/
        SIZE        fmbfsz ;    /* size of free message buffer area (in bytes,
                                   except the minimum control area) */
} T_RMBF ;
```

Packet format of rendezvous creation information

```
typedef struct t_cpor {
        ATR         poratr ;    /* rendezvous port attribute*/
        UINT        maxcmsz ;   /* maximum calling message size (in bytes) */
        UINT        maxrmsz ;   /* maximum reply message size (in bytes) */
        VB const *  name ;      /* rendezvous port name */
} T_CPOR ;
```

Packet format of rendezvous port state

```
typedef struct t_rpor {
        ID          ctskid ;    /* ID number of the first task in the rendezvous
                                   port's call-wait queue*/
        ID          atskid ;    /* ID number of the first task in the rendezvous
                                   port's accept-wait queue*/
} T_RPOR ;
```

Packet format of rendezvous state

```
typedef struct t_rrdv {
        ID          wtskid ;    /* ID number of task in the rendezvous termination
                                   waiting state */
} T_RRDV ;
```

(5) Memory pool management functions

Packet format of fixed-length memory pool creation information

```
typedef struct t_cmpf {
        ATR         mpfatr ;    /* fixed-length memory pool attribute */
        UINT        blkcnt ;    /* number of memory blocks that can be
                                   acquired (quantity)*/
        UINT        blksz ;     /* memory block size (in bytes)
        VP          mpf ;       /* start address of the fixed-length memory pool
                                   area */
        VB const *  name ;      /* fixed-length memory pool name */
} T_CMPF ;
```

Packet format of fixed-length memory pool state

```
typedef struct t_rmpf {
        ID          wtskid ;    /* ID number of the first task in the fixed-length
                                   memory pool's wait queue*/
        UINT        fblkcnt ;   /* number of free memory blocks in the
                                   fixed-length memory pool (quantity)*/

} T_RMPF ;
```

Packet format of variable length memory pool creation information

```
typedef struct t_cmpl {
        ATR         mplatr ;    /* variable length memory pool attribute */
        SIZE        mplsz ;     /* size of the variable length memory pool area
                                   (in bytes) */
        VP          mpl ;       /* start address of the variable length memory
                                   pool area */
        VB const *  name ;      /* variable length memory pool name */
} T_CMPL ;
```

Packet format of variable length memory pool state

```
typedef struct t_rmpl {
        ID          wtskid ;    /* ID number of the first task in the variable
                                   length memory pool's wait queue*/
        SIZE        fmplsz ;    /* total size of free variable length memory pool
                                   area (in bytes) */
        UINT        fblksz ;    /* maximum memory block size which can be
                                   acquired instantly (in bytes) */
} T_RMPL ;
```

（6）Time management functions

Packet format of cyclic handler creation information

```
typedef struct t_ccyc {
        ATR         cycatr ;    /* cyclic handler attribute */
        VP_INT      exinf ;     /* cyclic handler extended information*/
        FP          cychdr ;    /* cyclic handler activate address*/
        RELTIM      cyctim ;    /* cyclic handler activation cycle */
        RELTIM      cycphs ;    /* cyclic handler activation phase */
        VB const *  name ;      /* cyclic handler name */
} T_CCYC ;
```

Packet format of cyclic handler state

```
typedef struct t_rcyc {
        STAT        cycstat ;   /* cyclic handler operational state */
        RELTIM      lefttim ;   /* time left before the next cyclic handler
                                   activation*/
} T_RCYC ;
```

Packet format of alarm handler creation information

```
typedef struct t_calm {
        ATR         almatr ;    /* alarm handler attribute */
        VP_INT      exinf ;     /* alarm handler extended information */
        FP          almhdr ;    /* alarm handler activate address */
        VB const *  name ;      /* alarm handler name */
} T_CALM ;
```

Packet format of alarm handler state

```
typedef struct t_ralm {
        STAT        almstat ;   /* alarm handler operational state*/
        RELTIM      lefttim ;   /* time left before the activation of the alarm
                                   handler */
} T_RALM ;
```

Packet format of overrun handler creation information

```
typedef struct t_dovr {
        ATR         ovratr ;    /* overrun handler attribute */
        FP          ovrhdr ;    /* overrun handler activate address */
} T_DOVR ;
```

Packet format of overrun handler state

```
typedef struct t_rovr {
        STAT        ovrstat ;   /* overrun handler operational state*/
        OVRTIM      leftotm ;   /* remaining processing time*/
} T_ROVR ;
```

（7）System state management functions

Packet format of system state

```
typedef struct t_rsys {
        SIZE    fsyssz ;    /* free system memory space (in bytes)*/
        SIZE    fstksz ;    /* free memory space for stack (in bytes)*/
        SIZE    fmplsz ;    /* free memory space for memory pool (in bytes)*/
        UH      utskid ;    /* number of already created task ID */
        UH      usemid ;    /* number of already created semaphore ID*/
        UH      uflgid ;    /* number of already created eventflag ID*/
        UH      udtqid ;    /* number of already created data queue ID*/
        UH      umbxid;     /* number of already created memory box ID*/
```

```
    UH      umtxid;     /* number of already created mutex ID*/
    UH      umbfid;     /* number of already created message buffer ID*/
    UH      uporid;     /* number of already created rendezvous port ID*/
    UH      umpfid;     /* number of already created fixed-length memory pool
                           ID*/
    UH      umplid;     /* number of already created variable length memory
                           pool ID*/
    UH      ualmid;     /* number of already created alarm handler ID*/
    UH      ucycid;     /* number of already created cyclic handler ID*/
    UH      uisrid;     /* number of already created interrupt service routine
                           ID*/
    UH      ssbcnt;     /* number of minimum SSB*/
} T_RSYS ;
```

(8) Interrupt management functions

Packet format of interrupt handler definition information

```
typedef struct t_dinh {
    ATR         inhatr ;    /* interrupt handler attribute */
    FP          inthdr ;    /* interrupt handler activate address */
    IMASK       imask ;     /* interrupt mask level of the interrupt handler */
} T_DINH ;
```

Packet format of interrupt service routine creation information

```
typedef struct t_cisr {
    ATR         isratr ;    /* interrupt service routine attribute */
    VP_INT      exinf ;     /* interrupt service routine extended information */
    INTNO       intno ;     /* interrupt number to which interrupt service routine
                               is to be attached*/
    FP          isr ;       /* interrupt service routine activate address*/
    IMASK       imask ;     /* interrupt mask level of the interrupt handler */
} T_CISR ;
```

Packet format of interrupt service routine state

```
typedef struct t_risr {
    INTNO       intno ;     /* interrupt number to which interrupt service
                               routine is to be attached*/
    FP          isr ;       /* interrupt service routine activate address*/
} T_RISR ;
```

（9）System configuration management functions

Packet format of configuration information

```
typedef struct t_rcfg {
        UH      tskpri_max ;    Maximum value of task priority
        UH      tskid_max ;     Maximum value of task ID
        UH      semid_max ;     Maximum value of semaphore ID
        UH      flgid_max ;     Maximum value of event flag ID
        UH      dtqid_max ;     Maximum value of data queue ID
        UH      mbxid_max ;     Maximum value of mailbox ID
        UH      mtxid_max ;     Maximum value of mutex ID
        UH      mbfid_max ;     Maximum value of message buffer ID
        UH      porid_max ;     Maximum value of rendezvous port ID
        UH      mpfid_max ;     Maximum value of fixed-length memory pool ID
        UH      mplid_max ;     Maximum value of variable length memory pool
                                ID
        UH      almid_max ;     Maximum value of alarm handler ID
        UH      cycid_max ;     Maximum value of cyclic handler ID
        UH      isrid_max ;     Maximum value of interrupt service routine ID
        UH      devid_max ;     Maximum value of device driver ID
        UH      tick ;          Time tick
        UH      ssb_cnt ;       Number of created system service block
} T_RCFG ;
```

Packet format of version information

```
typedef struct t_rver {
        UH          maker ;     /* kernel maker code */
        UH          prid ;      /* identification number of kernel*/
        UH          spver ;     /* version number of the ITRON specification*/
        UH          prver ;     /* version number of kernel*/
        UH          prno[4] ;   /* management information of kernel product */
} T_RVER ;
```

7.3 Constants and Macros

(1) General

NULL	0	Invalid pointer
TRUE	1	True
FALSE	0	False
E_OK	0	Successful completion

(2) Object attributes

TA_NULL	0	Object attribute unspecified
TA_HLNG	0x00	Start a processing unit through a high-level language interface
TA_ASM	0x01	Start a processing unit through an assembly language interface
TA_TFIFO	0x00	Task wait queue is in FIFO order
TA_TPRI	0x01	Task wait queue is in task priority order
TA_MFIFO	0x00	Message queue is in FIFO order
TA_MPRI	0x02	Message queue is in message priority order
TA_ACT	0x02	Task is activated after creation
TA_RSTR	0x04	Restricted task
TA_WSGL	0x00	Not allow multiple task waiting for eventflag
TA_WMUL	0x02	Allow multiple task waiting for eventflag
TA_CLR	0x04	Clear eventflag when released from waiting
TA_INHERIT	0x02	Support the priority inheritance protocol for mutex
TA_CEILING	0x03	Support the priority ceiling protocol for mutex
TA_STA	0x02	Cyclic handler is in operational state after creation
TA_PHS	0x04	Preserving the cyclic handler phase

(3) Timeout specification

TMO_POL	0	Polling
TMO_FEVR	−1	Waiting forever
TMO_NBLK	−2	Non-blocking

（4）Service call operational mode

TWF_ANDW	0x00	AND waiting for eventflag
TWF_ORW	0x01	OR waiting for eventflag

（5）Object states

TTS_RUN	0x01	RUNNING state
TTS_RDY	0x02	READY state
TTS_WAI	0x04	WAITING state
TTS_SUS	0x08	SUSPENDED state
TTS_WAS	0x0c	WAITING-SUSPENDED state
TTS_DMT	0x10	DORMANT state
TTW_SLP	0x0001	Sleeping state
TTW_DLY	0x0002	Delayed state
TTW_SEM	0x0004	Waiting state for semaphore resource acquisition
TTW_FLG	0x0008	Waiting state for the eventflag
TTW_SDTQ	0x0010	Sending waiting state for a data queue
TTW_RDTQ	0x0020	Receiving waiting state for a data queue
TTW_MBX	0x0040	Receiving waiting state for a mailbox
TTW_MTX	0x0080	Waiting state for locking mutex
TTW_SMBF	0x0100	Sending waiting state for a message buffer
TTW_RMBF	0x0200	Receiving waiting state for a message buffer
TTW_CAL	0x0400	Calling waiting state for a rendezvous
TTW_ACP	0x0800	Accepting waiting state for a rendezvous
TTW_RDV	0x1000	Terminating waiting state for a rendezvous
TTW_MPF	0x2000	Acquiring waiting state for a fixed-length memory block
TTW_MPL	0x4000	Acquiring waiting state for a variable-length memory block
TTEX_ENA	0x00	Task exception handling enabled state
TTEX_DIS	0x01	Task exception handling disabled state
TCYC_STP	0x00	Cyclic handler is in non-operational state
TCYC_STA	0x01	Cyclic handler is in operational state
TALM_STP	0x00	Alarm handler is in non-operational state
TALM_STA	0x01	Alarm handler is in operational state
TOVR_STP	0x00	Processing time limit is not set
TOVR_STA	0x01	Processing time limit is set

（6）Other constants

TSK_SELF	0	Specifying invoking task
TSK_NONE	0	No applicable task
TPRI_SELF	0	Specifying the base priority of the invoking task
TPRI_INI	0	Specifying the initial priority of the task

7.4 Configuration constants and Marcos

（1）Priofity range

TMIN_TPRI	Minimum task priority (＝1)
TMAX_TPRI	Maximum task priority (＝31)
TMIN_MPRI	Minimum message priority (＝1)
TMAX_MPRI	Maximum message priority (＝31)

（2）Version information

TKERNEL_MAKER	Kernel maker code
TKERNEL_PRID	Identification number of the kernel
TKERNEL_SPVER	Version number of the ITRON specification
TKERNEL_PRVER	Version number of the kernel

（3）Maximum queuing /nesting count

TMAX_ACTCNT	Maximum activation request queuing count of the task (＝999)
TMAX_WUPCNT	Maximum wakeup request queuing count of the task (＝999)
TMAX_SUSCNT	Maximum suspension request nesting count of the task (＝999)

（4）Number of bits in bit pattern

TBIT_TEXPTN	Number of bits in the task exception reason
TBIT_FLGPTN	Number of bits in the eventflag
TBIT_RDVPTN	Nmber of bits in the rendezvous condition.

（5）Required memory area size

SIZE dtqsz = TSZ_DTQ (UINT dtqcnt)

Size of data queue area necessary store dtqcnt data (in bytes)

SIZE mprihdsz = TSZ_MPRIHD (PRI maxmpri)

Size of the message queue header area for each priority necessary for mailbox with the maximum message priority of the messages to be sent is maxmpri (bytes)

SIZE mbfsz = TSZ_MBF (UINT msgcnt, UINT msgsz)

Size of the message buffer area necessary to buffer msgcnt messages each of size msgsz byte (approximate bytes)

SIZE mpfsz = TSZ_MPF (UINT blkcnt, UINT blksz)

Size of the fixed-length memory pool area necessary to be able to acquire blkcnt memory blocks each of size blksz bytes (in bytes)

SIZE mplsz = TSZ_MPL (UINT blkcnt, UINT blksz)

Size of the variable length memory pool area necessary to be able to acquire blkcnt memory blocks each of size blksz bytes (approximate bytes)

（6）Others

TMAX_MAXSEM	Maximum value of the maximum semaphore resource count (＝999)

7.5 Error code list

E_SYS	-5	0xFFFFFFFB	System error
E_NOSPT	-9	0xFFFFFFF7	Unsupported function
E_RSFN	-10	0xFFFFFFF6	Reserved function code
E_RSATR	-11	0xFFFFFFF5	Reserved attribute
E_PAR	-17	0xFFFFFFEF	Parameter error
E_ID	-18	0xFFFFFFEE	Invalid ID number
E_CTX	-25	0xFFFFFFE7	Context error
E_MACV	-26	0xFFFFFFE6	Memory access violation
E_OACV	-27	0xFFFFFFE5	Object access violation
E_ILUSE	-28	0xFFFFFFE4	Illegal service call use
E_NOMEM	-33	0xFFFFFFDF	Insufficient memory
E_NOID	-34	0xFFFFFFDE	Lack of ID number
E_OBJ	-41	0xFFFFFFD7	Object state error
E_NOEXS	-42	0xFFFFFFD6	Non-existent object
E_QOVR	-43	0xFFFFFFD5	Queuing overflow
E_RLWAI	-49	0xFFFFFFCF	Forced release from waiting
E_TMOUT	-50	0xFFFFFFCE	Polling failure or timeout
E_DLT	-51	0xFFFFFFCD	Waiting object deleted
E_CLS	-52	0xFFFFFFCC	Waiting object state changed
E_WBLK	-57	0xFFFFFFC7	Non-blocking accepted
E_BOVR	-58	0xFFFFFFC6	Buffer overflow

7.6 System call list

System call name	Task	Initialization handler	Time event handler	Interrupt service routine
A）Task management functions				
cre_tsk／acre_tsk	○	○	×	×
del_tsk	○	×	×	×
act_tsk／iact_tsk	○	○	○	○
can_act	○	○	○	×
sta_tsk	○	○	○	○
ext_tsk	○	×	×	×
exd_tsk	○	×	×	×
ter_tsk	○	×	×	×
chg_pri	○	○	○	×
get_pri	○	○	○	×
ref_tsk	○	○	○	×
ref_tst	○	○	○	×
B) Task dependent synchronization				
slp_tsk	○	×	×	×
tslp_tsk	○	×	×	×
wup_tsk／iwup_tsk	○	○	○	○
can_wup	○	○	○	×
rel_wai／irel_wai	○	○	○	○
sus_tsk	○	○	○	×
rsm_tsk	○	○	○	×
frsm_tsk	○	○	○	×
dly_tsk	○	×	×	×

System call name	Task	Initialization handler	Time event handler	Interrupt service routine
C) Task exception handling				
def_tex	×	×	×	×
ras_tex	×	×	×	×
iras_tex	×	×	×	×
dis_tex	×	×	×	×
ena_tex	×	×	×	×
sns_tex	×	×	×	×
ref_tex	×	×	×	×
D) Synchronization and Communication Semaphore				
cre_sem／acre_sem	○	○	×	×
del_sem	○	×	×	×
sig_sem／isig_sem	○	○	○	○
wai_sem	○	×	×	×
pol_sem	○	○	○	×
twai_sem	○	×	×	×
ref_sem	○	○	○	×
E) Synchronization and Communication Eventflag				
cre_flg／acre_flg	○	○	×	×
del_flg	○	×	×	×
set_flg／iset_flg	○	○	○	○
clr_flg	○	○	○	×
wai_flg	○	×	×	×
pol_flg	○	○	○	×
twai_flg	○	×	×	×
ref_flg	○	○	○	×

System call name	Task	Initialization handler	Time event handler	Interrupt service routine
F) Synchronization and Communication　Data queue				
cre_dtq／acre_dtq	○	○	×	×
del_dtq	○	×	×	×
snd_dtq	○	×	×	×
psnd_dtq／ipsnd_dtq	○	○	○	○
tsnd_dtq	○	×	×	×
fsnd_dtq／ifsnd_dtq	○	○	○	○
rcv_dtq	○	○	○	×
prcv_dtq	○	○	○	×
trcv_dtq	○	×	×	×
ref_dtq	○	○	○	×
G)　Synchronization and Communication　Mail box				
cre_mbx／acre_mbx	○	○	×	×
del_mbx	○	×	×	×
snd_mbx	○	○	○	×
rcv_mbx	○	×	×	×
prcv_mbx	○	○	○	×
trcv_mbx	○	×	×	×
ref_mbx	○	○	○	×
H)　Extended synchronization and communication　Mutex				
cre_mtx／acre_mtx	○	○	×	×
del_mtx	○	×	×	×
loc_mtx	○	×	×	×
ploc_mtx	○	×	×	×
tloc_mtx	○	×	×	×
unl_mtx	○	×	×	×
ref_mtx	○	○	○	×

System call name	Task	Initialization handler	Time event handler	Interrupt service routine
I) Extended synchronization and communication Message buffer				
cre_mbf／acre_mbf	○	○	×	×
del_mbf	○	×	×	×
snd_mbf	○	×	×	×
psnd_mbf	○	○	○	×
tsnd_mbf	○	×	×	×
rcv_mbf	○	×	×	×
prcv_mbf	○	○	○	×
trcv_mbf	○	×	×	×
ref_mbf	○	○	○	×
J) Extended synchronization and communication Rendezvous				
cre_por／acre_por	○	○	×	×
del_por	○	×	×	×
cal_por	○	×	×	×
tcal_por	○	×	×	×
acp_por	○	×	×	×
pacp_por	○	×	×	×
tacp_por	○	×	×	×
fwd_por	○	×	×	×
rpl_rdv	○	×	×	×
ref_por	○	○	○	×
ref_rdv	○	○	○	×
K）Memory pool management Fixed-length memory pool				
cre_mpf／acre_mpf	○	○	×	×
del_mpf	○	×	×	×
get_mpf	○	×	×	×
pget_mpf	○	○	○	×
tget_mpf	○	×	×	×
rel_mpf	○	○	○	×
ref_mpf	○	○	○	×

System call name	Task	Initialization handler	Time event handler	Interrupt service routine
L） Memory pool management　Variable length memory pool				
cre_mpl／acre_mpl	○	○	×	×
del_mpl	○	×	×	×
get_mpl	○	×	×	×
pget_mpl	○	○	○	×
tget_mpl	○	×	×	×
rel_mpl	○	○	○	×
ref_mpl	○	○	○	×
M) Time management　System time management				
set_tim	○	○	○	×
get_tim	○	○	○	×
isig_tim	×	×	×	○
N） Time manaegement　Cylic handler				
cre_cyc／acre_cyc	○	○	×	×
del_cyc	○	×	×	×
sta_cyc	○	○	○	×
stp_cyc	○	○	○	×
ref_cyc	○	○	○	×
O） Time manaegement　Alarm handler				
cre_alm／acre_alm	○	○	×	×
del_alm	○	×	×	×
sta_alm	○	○	○	×
stp_alm	○	○	○	×
ref_alm	○	○	○	×

System call name	Task	Initialization handler	Time event handler	Interrupt service routine
P）Time manaegement　Overrun handler				
def_ovr	○	○	×	×
ivsig_ovr	×	×	×	○
sta_ovr	○	○	○	×
stp_ovr	○	○	○	×
ref_ovr	○	○	○	×
Q）System state management				
rot_rdq／irot_rdq	○	○	○	○
get_tid／iget_tid	○	○	○	○
loc_cpu／iloc_cpu	○	○	○	○
unl_cpu／iunl_cpu	○	○	○	○
dis_dsp	○	×	×	×
ena_dsp	○	×	×	×
sns_ctx	○	○	○	○
sns_loc	○	○	○	○
sns_dsp	○	○	○	○
sns_dpn	○	○	○	○
ref_sys	○	○	○	×
R）Interrupt management				
def_inh	○	○	○	×
cre_isr／acre_isr	○	○	×	×
del_isr	○	×	×	×
ref_isr	○	○	○	×
dis_int	△	△	△	△
ena_int	△	△	△	△
chg_ims	○	○	○	○
get_ims	○	○	○	○
S）Service call management functions				
def_svc	×	×	×	×
cal_svc	×	×	×	×

System call name	Task	Initialization handler	Time event handler	Interrupt service routine
T）System configuration management functions				
def_exc	△	△	△	△
ref_cfg	○	○	○	○
ref_ver	○	○	○	○
U）Peculiar functions ・Device driver management function				
vdef_dev	○	○	×	×
vctr_dev	○	×	×	×

○：Usable

×：Unusable

△：Depending on the processor

Index

E

F

G

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

µC3/Standard Users Guide

2008	September	1st Edition
2008	November	2nd Edition
2010	June	3rd Edition

eForce Co., Ltd. http://www.eforce.co.jp/

7F Genbei Building,5-4 Nihonbashi Tomizawa-cho,Chuo-ku,Tokyo Japan, 103-0006

TEL 03-5614-6918 FAX 03-5614-6919

Contact us : info@eforce.co.jp