TOPPERS/ASP Kernel Specification

July 2016
Revision 1.0.0

A.I.Corporation

NAGOYA UNIVERSITY

NCES

Center for Embedded Computing Systems

Disclaimer

This document has been provided only for TOPPERS APPLICATION CONTEST (http://toppers.jp/contest.html#2016) and GR DESIGN CONTEST 2016 (http://gadget.renesas.com/en/). It is not allowed to use or redistribute for any other purposes. If you want to use or redistribute for any other purposes, contact us please.

Revision History

Date	Rev.	Status	Applicable version of ASP kernel	Change
07/12/2016	1.0.0	Release	1.9.2	Initial release.

CONTENT

1. INTRODUCTION 8
   1.1. Intended Readers 8
   1.2. References: 8
2. ASP KERNEL OVERVIEW 9
   2.1. Basic concepts of ASP Kernel 9
      2.1.1. Hardware requirements 9
      2.1.2. System State 10
      2.1.3. System time and processes dependent thereon 11
      2.1.4. CPU exception handler 11
      2.1.5. Interrupt handler / interrupt service routine 11
      2.1.6. Task scheduling 11
   2.2. Advanced Standard Profile 15
      2.2.1. Waiting for multiple tasks of event flag 15
      2.2.2. Cyclic Handler 15
      2.2.3. Alarm Handler 16
   2.3. Changes from µITRON 4.0 Specification 16
      2.3.1. Review of ITRON standard data type 16
      2.3.2. Call of ext_tsk from a non-task context. 16
      2.3.3. Additional operations that can be used in CPU exception handler 16
      2.3.4. Adoption of some µITRON Protection eXtensions (PX) 18
      2.3.5. Review of data types such as process unit 18
      2.3.6. Review of constants with the value 0 (including object attribute) 18
      2.3.7. Abolishment of forced wait request nest 19
      2.3.8. Abolishment of system time setting function 19
      2.3.9. Modification to Cyclic Handler Specification 19
      2.3.10. Change of execution start condition of task exception handling routine 19
      2.3.11. Interrupt Service Routine 20
      2.3.12. Reference object state 21
   2.4. Change regarding Proprietary Expansion Function of TOPPERS/JSP 22
      2.4.1. System time reference function for performance evaluation 22
      2.4.2. System termination process routine function 22
      2.4.3. Reference kernel inactive state 22
   2.5. Function Expansion Unique to TOPPERS New Generation Kernel 23
      2.5.1. Configuration of attribute of interrupt request line 23
      2.5.2. Re-initialisation function of synchronisation and communication object 23
      2.5.3. New introduction of priority data queue 23
      2.5.4. Reference expansion information of own task 24
      2.5.5. Termination of kernel 24
      2.5.6. Configuration of stack area for non-task context 24
   2.6. Review of System Configuration Specification 25
      2.6.1. Modification of handling of INCLUDE static API and pre-processor directive in

C-language 25
3. FIXED PARMETERS OF TOPPERS/ASP 26
3.1. Fixed Parameters of Kernel 26
3.2. Interrupt Number 26
3.3. Interrupt Handler Number 26
3.4. CPU Exception Handler Number 27
3.5. API Naming Rule 27
3.6. TOPPERS Common Data Type 27
3.7. Definition of Configuration File Description Specification 28
3.7.1. Regarding configuration file and static API 28
3.7.2. Types of static API parameters 33
4. STATIC API REFERENCE 35
4.1. Inclusion of INCLUDE configuration file 35
4.2. CRE_TSK Create Task 36
4.3. DEF_TEX Define Task Exception Handling Routine 38
4.4. CRE_SEM Create Semaphore 39
4.5. CRE_FLG Create Event Flag 40
4.6. CRE_DTQ Create Data Queue 41
4.7. CRE_PDQ Create Priority Data Queue 42
4.8. CRE_MBX Create Mailbox 44
4.9. CRE_MFP Create Fixed-length Memory Pool 45
4.10. CRE_CYC Create Cyclic Handler 47
4.11. CRE_ALM Create Alarm Handler 49
4.12. CFG_INT Configure Interrupt Request Line Attribute 50
4.13. DEF_INH Define Interrupt Handler 52
4.14. DEF_EXC Define CPU Exception Handler 54
4.15. DEF_ICS Set Stack Area for Non-Task Context 55
4.16. ATT_ISR Add Interrupt Service Routine 56
4.17. ATT_INI Add Initialised Routine 58
4.18. ATT_TER Add Termination Process Routine 59
5. KERNEL DYNAMIC APIs (SERVICE CALL) REFERENCES 60
5.1. Header File 60
5.2. Task Management Function 61
5.2.1. act_tsk/iact_tsk Activate Task 61
5.2.2. can_act Cancel task activation request 63
5.2.3. ext_tsk Terminate Own Task 64
5.2.4. ter_tsk Forcibly Terminate Other Tasks 66
5.2.5. get_pri Get Current Priority of Task 67
5.2.6. chg_pri Change Base Priority of Task 68
5.2.7. get_inf Reference Extension Information of Own Task 69
5.2.8. ref_tsk Reference Task State 70
5.3. Task-associated Synchronisation Function 73
5.3.1. slp_tsk/tslp_tsk Wait for Task to Wake-up 73

5.3.2. wup_tsk/iwup_tsk Wake Up Task 74
5.3.3. can_wup Cancel Task Wake-up Request 75
5.3.4. rel_wai/irel_wai Forcibly Release from Task-Waiting State 76
5.3.5. sus_tsk Suspend Task 77
5.3.6. rsm_tsk Resume Task From Suspended State 78
5.3.7. frsm_tsk Forcibly resume task from suspended state (Abolished) 78
5.3.8. dly_tsk Delay Execution of Own Task 79
5.4. Task Exception Handling Function 80
5.4.1. ras_tex/iras_tex Raise Task Exception Handling 80
5.4.2. dis_tex Disable Task Exception Handling 81
5.4.3. ena_tex Enable Task Exception Handling 82
5.4.4. sns_tex Reference Task Exception Handling Disabled State 83
5.4.5. ref_tex Reference Task Exception Handling State 84
5.5. Semaphore Functions 85
5.5.1. sig_sem/isig_sem Return Semaphore Resource 85
5.5.2. wai_sem/pol_sem/twai_sem Aquire Semaphore Resource 86
5.5.3. ini_sem Re-initialise Semaphore 87
5.5.4. ref_sem Reference Semaphore State 88
5.6. Event Flag Function 89
5.6.1. set_flg/iset_flg Set Event Flag 89
5.6.2. clr_flg Clear Event Flag 90
5.6.3. wai_flg/pol_flg/twai_flg Wait for Event Flag 91
5.6.4. ini_flg Re-initialise Event Flag 93
5.6.5. ref_flg Reference Event Flag State 94
5.7. Data Queue Function 95
5.7.1. snd_dtq/psnd_dtq/ipsnd_dtq/tsnd_dtq Send to Data Queue 95
5.7.2. fsnd_dtq/ifsnd_dtq Forcibly Send to Data Queue 97
5.7.3. rcv_dtq/prcv_dtq/trcv_dtq Receive from Data Queue 98
5.7.4. ini_dtq Re-initialise Data Queue 99
5.7.5. ref_dtq Reference Data Queue State 100
5.8. Priority Data Queue Function 101
5.8.1. snd_pdq/psnd_pdq/tsnd_pdq/ipsnd_pdq Send to Priority Data Queue 101
5.8.2. rcv_pdq/prcv_pdq/trcv_pdq Receive from Priority Data Queue 104
5.8.3. ini_pdq Re-initialise Priority Data Queue 106
5.8.4. Reference Priority Data Queue State 107
5.9. Mailbox Function 108
5.9.1. snd_mbx Send to Mailbox 108
5.9.2. rcv_mbx/prcv_mbx/trcv_mbx Receive from Mailbox 109
5.9.3. ini_mbx Re-initialise Mailbox 111
5.9.4. ref_mbx Reference Mailbox State 112
5.9.5. Description of Message (T_MSG) Type 113
5.10. Fixed-Length Memory Pool Function 114
5.10.1. get_mpf/pget_mpf/tget_mpf Acquire Fixed-Length Memory Block 114

5.10.2. rel_mpf Release Fixed-Length Memory Block 116
5.10.3. ini_MPF Re-initialise Fixed-Length Memory Pool 117
5.10.4. ref_mpf Reference Fixed-Length Memory Pool State 118
5.11. Cyclic Handler Function 119
5.11.1. sta_cyc Start Cyclic Handler 119
5.11.2. stp_cyc Stop Cyclic Hander 121
5.11.3. ref_cyc Reference Cyclic Handler State 122
5.12. Alarm Handler Function 123
5.12.1. sta_alm/ista_alm Start Alarm Handler 123
5.12.2. stp_alm/istp_alm Stop Alarm Handler 124
5.12.3. ref_alm Reference Alarm Handler State 125
5.13. Time Management Function 126
5.13.1. get_tim Reference System Time 126
5.13.2. set_tim Set System Time (Abolished) 126
5.14. System State Management Function 127
5.14.1. get_tid/iget_tid Get Task ID under Execution 127
5.14.2. loc_cpu/iloc_cpu Transition to CPU Locked State 128
5.14.3. unl_cpu/iunl_cpu Unlock CPU Locked State 129
5.14.4. ena_dsp Enable Dispatch 130
5.14.5. dis_dsp Disable Dispatch 131
5.14.6. rot_rdq/irot_rdq Rotate Task Execution Order 132
5.14.7. sns_xxx Reference Different States 133
5.15. Interrupt Management Function 134
5.15.1. ena_int () Enable Interrupt 134
5.15.2. dis_int() Disable Interrupt 135
5.15.3. chg_ipm Change Interrupt Priority Mask Level 136
5.15.4. get_ipm Reference Interrupt Priority Mask Level 137

1. INTRODUCTION

TOPPERS/ASP kernel is an implementation of a real-time kernel for single-core architectures in compliance with the specification of TOPEERS new generation integrated kernel developed and maintained by TOPPERS Project, an NPO. The specification of the new generation integrated kernel has been drafted by the TOPPERS Project based on μITRON Ver 4.0 with the aim to expand and sophisticate μITRON Ver 4.0 specification formulated by TRON Forum.

ASP stands for Advanced Standard Profile, which was the reviewed set of the standard profile functions compliant with μITRON Ver 4.0.

This document defines the specification of the application programming interface (hereinafter referred to as "API"), of TOPPERS/ASP kernel, where any difference from the conventional specification of μITRON Ver 4.0 is expressly clarified.

1.1. Intended Readers

This document is meant for developers of a TOPPERS/ASP application and describes the API services of the kernel. TOPPERS/ASP is a kernel compliant with "TPPERS new generation kernel integration specification" published by TOPPERS Project. "TOPPERS new generation kernel integration specification" is recommended.

TOPPERS new generation kernel succeeds the functional concept and the specification of μITRON.

TOPPERS/ASP is meant for software developers having the knowledge of μITRON 4.0 Specification and concept and the general knowledge. Perusal of μITRON Kernel Specification, issued by TRON Forum defined as Reference.

1.2. References:

1. "μITRON Specification Ver 4.02.00" supervised by Ken Sakamura / edited by Hiroaki Takada published by ITRON Group of TRON Forum
2. "TOPPERS New Generation Kernel Integration Specification" published by TOPPERS Project
   - Retrievable from http://www.toppers.jp/documents.html#ngki_spec
   - The latest version for the time being is Release 1.5.0 (2012-12-26)
3. "Migration Guide to TOPPERS New Generation Kernel" published by TOPPERS Project
   - Retrieval from http://www.toppers.jp/documents.html#migration
   - The latest version for the time being is Release 1.2.0 (22/02/2010)
4. "TOPPERS/ASP Kernel User Manual" published by TOPPERS Project
   - Included in this package. (doc/ asp_docs /user.txt)
5. "TOPPERS/ASP Kernel Specification Overview" published by TOPPERS Project
   - Included in this package. (doc/asp_docs/asp_spec.txt)

2. ASP KERNEL OVERVIEW

TOPPERS/AMP kernel operates with the specification defined by the single-processor support functions provided in "TOPPERS New General Kernel Integration Specification". The scope of application of ASP kernel is a system configured with a simple -processor. Basically, the designer is to decide statically the processor that executes a task. At design phase (i.e. before system runtime), the designer also needs to declare how many kernel objects including tasks, semaphores, eventflags, etc. are used in the application.
Note that TOPPERS/ASP kernel does not have a dynamic heap memory management function as its implementation policy. Therefore, TOPPER/ASP kernel does not support kernel object creation or deletion function at runtime, either.

2.1. Basic concepts of ASP Kernel

2.1.1. Hardware requirements

ASP kernel assumes that it would be implement on embedded systems that meet the following hardware requirements.

1. At least one hardware timer can be used for system time management.
2. At least one serial device can be used for system logging function.

The target-dependent parts of ASP kernel provide device drivers for each device.

2.1.2. System State

In a multi-processor supporting kernel, many of system states are maintained per processor. In concrete, the states that are maintained per processor in the standard profile are as follows:

- Tasks in execution
- Kernel operation state and non-operation state
- All interrupts locked state and all interrupts released state
- CPU locked state and CPU unlocked state
- Interrupt priority mask
- Dispatch enabled status and dispatch disabled status
- Dispatch pending state
- Spin lock retrieved state

2.1.3. System time and processes dependent thereon

For the system time control, either the local timer scheme where it is controlled per processor or the global timer scheme where the single time is managed by the total system is selected.

2.1.4. CPU exception handler

CPU exception handler shall be configured per processor.
It is necessary to describe that the definition of a CPU exception handler (DEF_EXC) should belong to one of the classes.

2.1.5. Interrupt handler / interrupt service routine

Interrupt handler / interrupt service routine, shall be configured statically that it should be accepted by one of the processors.

2.1.6. Task scheduling

A task execution shall be allocated to one of the processors. Each processor shall perform priority-based scheduling to the task allocated thereto.

2.2. Advanced Standard Profile

The new generation kernel integration specification has introduced a set of functions not included in the standard profile of the subsequent µITRON 4.0 specification. This is referred to as Advanced Standard Profile. In the subsequent paragraphs, Advanced Standard Profile is described.

2.2.1. Waiting for multiple tasks of event flag

The standard profile of µITRON Ver. 4 provides that an event flag need not support the function of waiting for multiple tasks (event flag with TA_WMUL attribute).
TOPPER/ASP (Advanced Standard Profile), implemented compliant with TOPPERS New Generation Kernel Integration Specification, has implemented the function for an event flag to wait for multiple tasks (event flag with TA_WMUL attribute) as in-scope of its standard profile.

2.2.2. Cyclic Handler

Advanced Standard Profile of TOPPERS New Generation Kernel Integration Specification does not support the cyclic handler with TA_PHS attribute so is not supported by µITRON Ver 4.0 standard profile.

2.2.3. Alarm Handler

The standard profile of µITRON Ver 4.0 stipulates that the alarm hander function need not be supported.
TOPPERS/ASP (Advanced Standard Profile), implemented in compliance with TOPPERS New Generation Kernel Integration Specification, has implemented the function for the alarm handler as in-scope of its standard profile.
The static APIs and the service calls of the alarm handler supported are as follows:

CRE_ALM	Generation of alarm handler (static API)
sta_alm	Start of alarm handler
ista_alm	Start of alarm handler (for non-task context)
stp_alm	Stop of alarm hander
istp_alm	Stop of alarm handler (for non-task context)

2.3. Changes from µITRON 4.0 Specification

Reviewing µITRON 4.0 Specification, some changes have been made in New General Kernel Specification. The changes are described below:

2.3.1. Review of ITRON standard data type

We reviewed ITRON standard types according to the new ISO C99 specification, which was not existent when drafting µITRON Ver 4.0 Specification.
Integer types (such as B, H and W) with the fixed size are modified to have the names compliant with C99 (such as int8_t, int16_t and int32_t).
For an application that requires ITRON standard data type (such as B, H and W), the legacy itron.h shall be available.

2.3.2. Call of ext_tsk from a non-task context.

In µITRON Ver 4.0 Specification, ext_tsk shall be an API that is not returned from this function.
However, complying with this specification, if this function is called in an unexpected context or system state, an error cannot be returned. So, the user should be responsible for calling this function correctly.
Reviewing this fact, New Generation Kernel Specification provides that, if ext_tsk is called from an abnormal state or task context, an E_CTX error should be returned. Consequently, there is the possibility that ext_tsk is returned, the type of the return value of which shall be ER.

2.3.3. Additional operations that can be used in CPU exception handler

For the operations that can be performed in CPU exception handler, they shall be compliant with TOPPERS standard interrupt process model.
Regarding this matter, neither TOPEERS/JSP Kernel Specification nor μITRON 4.0 is adhered to.
Therefore, the following service calls have been implemented as service calls:

xsns_dpn Reference dispatch pending state when CPU exception occurs

xsbs_xpn Reference task exception handling pending state when CPU exception occurs

2.3.4. Adoption of some µITRON Protection eXtensions (PX)

µITRON 4.0\PX (Protection extensions) Specification enhanced the protection as modification to µITRON 4.0 Specification by separating the management memory region used by the kernel and the user.

Considering that part of this PX Specification is effective in advancing the safety of the kernels not subject to protection extensions, it has been adopted in New Generation Kernel Specification. The part of PX Specification adopted is explained in the subsequent paragraphs:

Fixed-length memory pool
A starting address for the fixed-length memory pool management block (mpfmb) has been added to a parameter for the static API that generates a fixed-length memory pool.

Data queue
To be consistent with the modified specification of fixed-length memory pool, the name of the last parameter of the static API (CRE_DTQ) that generates a queue has been changed to the starting address of data queue (dtqmb).

In PX Specification, the mailbox specification was also modified so that the management area and the user area of message are separated. However, if PX Specification was adopted, the compatibility of behaviour would be lost as it is differently specified in µITRON 4.0 Specification. So, this is not adopted in ASP Specification. There is no modification from µITRON Specification regarding the static API (CRE_MBX) that generates a mailbox.

2.3.5. Review of data types such as process unit

The data type of the entry address of process unit shall now be defined with the dedicated data type per process unit instead of PF type generalised so that it can be applied to address with all types. For example, the data type of the entry address of a task shall be TASK type and that of a task exception handling routine shall be TEXRTN type.
The data type of the stack area the fixed-length memory pool shall also be defined with STK_T type and MPF_T type respectively.

2.3.6. Review of constants with the value 0 (including object attribute)

To prevent a coding mistake, handling of the constants with the value 0 is reviewed. For the object attributes (such as TA_TFIFO), those with the value 0 will be handled as default and abolished, then the definition is moved to itron.h. For the service call behaviour mode (TWF_ANDW) and the object states (TTEX_ENA) with the value 0, the values will be changed.

2.3.7. Abolishment of forced wait request nest

The function that nests a forced wait request is abolished. In other words, the maximum value of the number of the forced wait request nests shall be fixed to 1. Consequently, frsm_tsk is abolished and frsm_tsk is defined as macro in rsm_tsk in itron.h. Further, TMAX_SUSCNT definition has been moved to itron.h.

2.3.8. Abolishment of system time setting function

set_tim (Set System Time) has been abolished.
In ASP, the system time cannot be set from a user program.
The reason for the abolishment is that there is a problem that, if changing the system time is allowed, many programs referring to the system time that is ignorant of such a change do not operate as expected. On the other hand, there is no advantage for the time being of allowing any operation that changes the system time from a user program, it was decided to abolish this function.

2.3.9. Modification to Cyclic Handler Specification

The time when the cyclic handler is activated for the first time after calling sta_cyc has been changed.
While, in µITRON 4.0 Specification, it is specified as the relative time specified with the activation cycle of the cyclic handler after calling sta_cyc, the same shall be the relative time specified with the activation phase in ASP kernel.

2.3.10. Change of execution start condition of task exception handling routine

In New Generation Kernel Specification, the condition has been changed so that the task exception handling routine shall not be initiated unless all the interrupt priority masks are released.

2.3.11. Interrupt Service Routine

μITRON Ver 4.0 Standard Profile stipulates that either an interrupt handler or an interrupt service routine may be supported.

Advanced Standard Profile of TOPPERS New Generation Kernel Integration Specification stipulates that, in addition to interrupt handlers, service routines should be supported. The additional static APIs are as follows:

ATT_ISR　Addition of interrupt service routine (static API)

2.3.12. Reference object state

Mainly, as the function for debugging, the function of reference to object state is implemented.

ref_tsk Reference task state
ref_tex Reference task exception handling
ref_sem Reference semaphore state
ref_flg Reference event flat state
ref_dtq Reference data queue state
ref_mbx Reference mailbox state
ref_mpf Reference fixed-length memory pool
ref_cyc Reference cyclic handler state
ref_alm Reference alarm handler state

2.4. Change regarding Proprietary Expansion Function of TOPPERS/JSP

The proprietary functions other than specified in μITRON Specification that are implemented in TOPPERS/JSP kernel have been reviewed in New Generation Kernel Specification and adopted as the standard functions. Such functions are described in the subsequent paragraphs:

2.4.1. System time reference function for performance evaluation

This function refers the system time for performance evaluation. It reads the system time in μ-seconds to measure the performance of the tasks running on the kernel or the kernel itself. Service call for this function is as follows:

get_utm Reference system time for performance evaluation

This is an API that was implemented as vxget_tim in JSP kernel.

As a modified part from JSP Kernel Specification, get_utm shall be called from any given context. Also, the type of the system time for performance evaluation (SYSUTM) is defined to ulong_t in the target non-dependent part.

2.4.2. System termination process routine function

New General Kernel supports the function to register the termination process routine that is called at the system termination. The static API for this function is as follows:

ATT_TER Addition of termination process routine (static API)

2.4.3. Reference kernel inactive state

If there is the possibility that a function called from a task operating on the kernel is called before the completion of the kernel initialisation or after the start of the termination process, it is necessary to determine if the service call of the kernel can be called therein. The service call for this function is as follows (the name has been changed from that of JSP Kernel)

sns_ker Reference inactive state of kernel

2.5. Function Expansion Unique to TOPPERS New Generation Kernel

2.5.1. Configuration of attribute of interrupt request line

As the function to configure the attribute of an interrupt request line, the following static API specified in TOPPERS standard interrupt process model.

CFG_INT	Configuration of attribute of interrupt request line

2.5.2. Re-initialisation function of synchronisation and communication object

The following service calls are implemented as the functions to return synchronisation and communication objects to the initialised state.

ini_sem	Re-initialisation of semaphore
ini_flg	Re-initialisation of event flag
ini_dtq	Re-initialisation of data queue
ini_mbx	Re-initialisation of mailbox
ini_mpf	Re-initialisation of fixed-length memory pool
ini_pdq	Re-initialisation of priority data queue

When a synchronisation or communication object is re-initialised, all the tasks in waiting for the synchronisation or communication object will be released from waiting state. E_DLT shall be returned to the tasks that are released from waiting state. E_DLT is an error code not used in the standard profile.

Therefore, more than one task may be waked up by these service calls. If many tasks are released from waiting state simultaneously, the period of time of disabling interrupts in the kernel becomes long. So, you have to be careful when using.

The application side is responsible for maintaining the consistency therewith when re-initialising a synchronisation or communication object. In concrete, when a fixed-length memory pool is re-initialised, the memory blocks acquired shall not be used from that point. Also, when re-initialising a mailbox, the message area that is sent to the mailbox shall be recollected.

2.5.3. New introduction of priority data queue

The function of data queue with priority has been newly introduced. The static APIS and the service calls to be implemented for the function of data queue with priority are as follows:

CRE_PDQ	Create priority data queue
snd_pdq	Send to priority data queue
psnd_pdq	Send to priority data queue (polling)
ipsnd_pdq	Send to priority data queue (pooling, non-task context)
tsnd_pdq	Send to priority data queue (with timeout)
ecv_pdq	Receive from priority data queue
prcv_pdq	Receive from priority data queue (polling)
trcv_pdq	Receive from priority data queue (with timeout)

ini_pdq Re-initialise priority data queue
ref_pdq Reference priority data queue state

2.5.4. Reference expansion information of own task

A service call to reference the expansion information of own task has been newly introduced. It is assumed to be used in the library, etc.

get_inf Reference expansion information of own task

2.5.5. Termination of kernel

A service call that can terminate the kernel in case a fatal error occurs has been newly introduced. This function is classified into the system configuration management function according to the function classification of μITRON 4.0 Specification (JSP kernel had the function substantially same as this with the name kernel_exit).

ext_ker Exit kernel

A kernel that does not have the protection function of New Generation Kernel will not be returned with this service call, but a kernel that has the protection function may be returned with an error. So the return value is fixed to ER.

Thus, kernel_start of JSP kernel is changed to sta_ker. However, sta_ker shall not be regarded as a service call.

2.5.6. Configuration of stack area for non-task context

A static API that specifies the starting address and the size of the stack area for non-task context has been newly introduced. It is also possible to specify, using this static API, only the size of the stack area for non-task context.

DEF_ICS Define stack area for non-task context

2.6. Review of System Configuration Specification

The system configuration processes were reviewed comprehensively. As a result, the handling of pre-processor directive of C-language and the classification of static API parameters in the system configuration file have been modified.

2.6.1. Modification of handling of INCLUDE static API and pre-processor directive in C-language

In µITRON Specification, INCLUDE static API was a static API in which INCLUDE of the header file defined in the kernel configuration and the initialisation file (kernel_cfg.c) was created and specified in the pre-processor directive format in C-language (#include ･･･) .
In ASP, an identical directive (#include) as the include directive (#include) of C-language pre-processor included in the system configuration file is created in the kernel configuration and initialisation file (kernel_cfg.c).
The order of the directives to be created is consistent with the order of the include directives in the system configuration file.
On the other hand, INCLUDE static API is not limited to the use of incorporating the configuration file (*.cfg) for comprising external definition. With this specification modification, only static APIs can be described in the file defined with INCLUDE static API.

3. FIXED PARMETERS OF TOPPERS/ASP

This chapter describes the fixed operating parameters of ASP.

3.1. Fixed Parameters of Kernel

Parameters implemented as the fixed parameters in the kernel internal are described hereunder.

Table 3.1 Kernel function fixed operating parameters

Kernel objects	Item	Value
Task	Number of activation request queues	1 The number of the suspended activation requests (act_tsk) per task shall be 1 at maximum.
	Number of wake-up request queues	1 The number of the suspended wake-up requests (wup_tsk) per task shall be 1 at maximum.
	The number forced waiting request nests	0 A forced waiting request is not suspended.
	The range of tasks priority	1～16
Mailbox	The range of message priority	1～16
Priority data queue	Data priority	1～16
Interrupt hander	Priority of interrupt line	CPU-dependent See the manual of the dependent part.
Interrupt Service Routine	The execution priority of interrupt service routine	1～16

3.2. Interrupt Number

CPU-dependent
See the manual of the dependent part.

3.3. Interrupt Handler Number

CPU-dependent
See the manual of the dependent part.

3.4. CPU Exception Handler Number

CPU-dependent
See the manual of the dependent part.

3.5. API Naming Rule

The static and dynamic APIs originated from µITRON Specification inherit the same.
For the newly introduced APIs, similar affinitive names are employed.

3.6. TOPPERS Common Data Type

With the change to TOPPERS common data type, the data type of the parameters of µITRON Specification has been changed.

INT → int_t
UINT → uint_t
VP → void *
VP_INT → intptr_t
BOOL → bool_t

TOPPERS common data type is defined in t_stddef.h It is incorporated by including kernel.h. t_stddef.h does not include any type of µITRON 4 Specification that was abolished.
Mainly, for reuse of µITRON 4 Specification RTOS (e.g.: TOPPERS/JSP) applications, some definitions remain in itron.h for those need the above-mentioned conventional ITRON types. Include this in case you need the abolished ITRON type.

3.7. Definition of Configuration File Description Specification

3.7.1. Regarding configuration file and static API

The format of the element unit that is managed by the kernel and statically incorporated in the kernel (hereinafter called "kernel object") as well as the framework incorporated into the kernel were standardised in μITRON Ver 4.0.
"TOPPERS New Generation Kernel Integration Specification" inherits this.
As the kernel objects, the format in which the kernel objects corresponding CPU, the system time, the task and the semaphore are defined is called a static API and the file in which static APIs are described is called a configuration file. A configuration file has .cfg extension idiomatically. The kernel tool that analyses the configuration file and generates the kernel management area including the task management block (TCB), etc. and its initialised information is called a configurator.

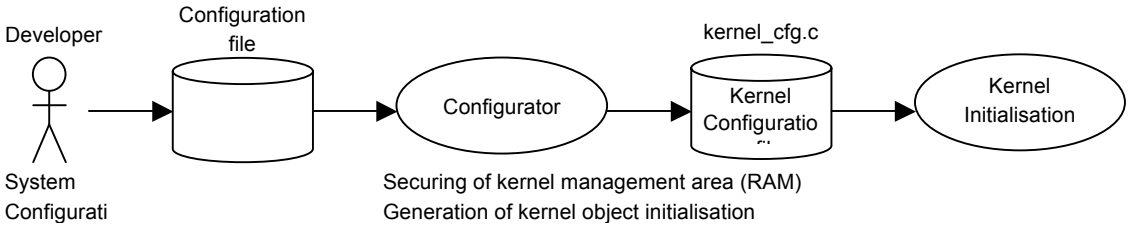

Figure 3.1 Concept of configurator

Before TOPPERS New Generation Kernel (JSP, FI4, IIMP), the syntax analysis and the output file generator function of a static API was incorporated in the source code of the configurator.
In TOPPERS New Generation Kernel, the syntax analysis and the output file generator function of a static API can be described externally and the output file generator function can be described in TOPPERS macro-processor macro language. The figure below depicts the relationship between input files and output files to/from the configurator:

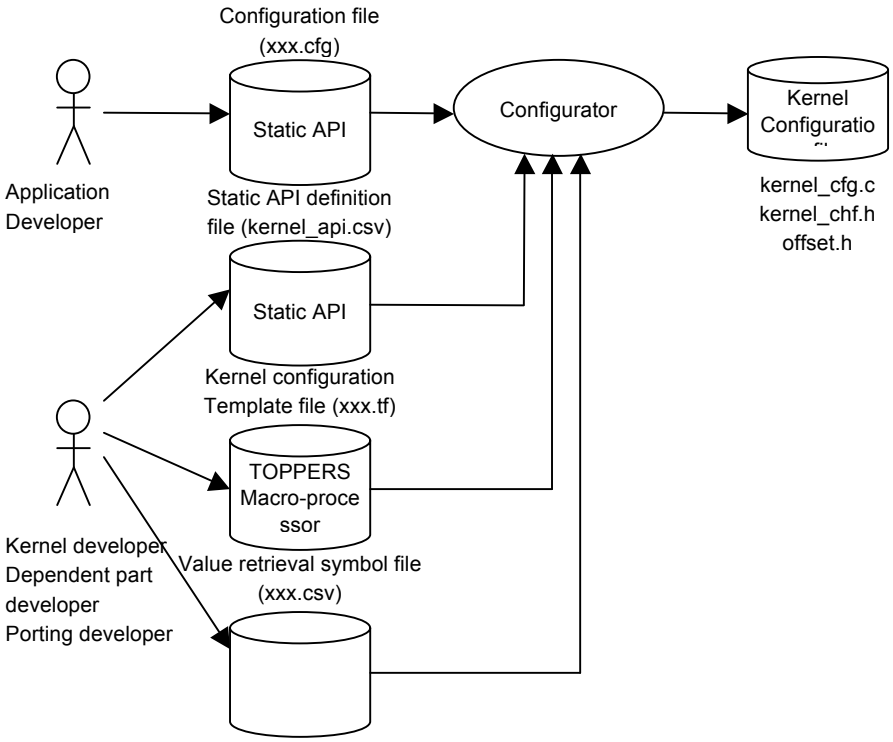

Figure 3.2　Output files to configurator

The input files for configuration control depicted above are described hereunder:

・Static API definition file

It is a file that defines the name and the syntax of a static API to be processed by the computer.

The static APIs supported by TOPPERS/ASP are available with the name kernel_api/cvc under kernel directory. The static API definition file is specified with --api-table object, which is a configurator command. It is also possible to create a file defined with an extension static API by the system uniquely. More than one static API definition file can be specified in a configurator.

・Value retrieval symbol file

In case use of the value of the assessment result of a constant expression in C-language is desired in a template file, it is necessary to specify a value retrieval symbol table. The value retrieval symbol table specifies --cfg1-def-table option as an argument to the configurator command. More than one value retrieval symbol table can be specified.

・Kernel configuration template file

It is a file describing the information generation format to a generation file in TOPPERS macro-processor macro language. As the macro-processor macro language can not only handle constants, variables and assignment statements but also describe selection control and repetitive control. The specification of this language is provided in "Specification of Configurator Built-in Macro-Processor for TOPPERS New Generation Kernel". A template file

is specified with ---template-file or -T=filename as an argument of the configurator command.

A template file can include other template files with the use of include syntax.
The hierarchy structure is provided below:

Template file for kernel_cfg.c creation
target\<target_system_dir>\target.tf
└ arch\<target_arch_dir>\<target_chip_dir>\<target_chip_name>.tf
└arch\<target_arch_dir>\<target_chip_dir>\chip.tf
└arch\<target_arch_dir>\common\core.tf
└kernel\kernel.tf

Template file of check process of kernel_cfg.c created
target\<target_system_dir>\target_check.tf
└ arch\<target_arch_dir>\<target_chip_dir>\chip_check.tf
└arch\<target_arch_dir>\common\core_check.tf
└kernel\kernel_check.tf

Template file for offset.h creation
arch\<target_arch_dir>\<target_chip_dir>\chip_offset.tf
└arch\<target_arch_dir>\common\core_offset.tf
└kernel\genoffset.tf

The following figure indicates the execution timing of a configurator in the total build process:

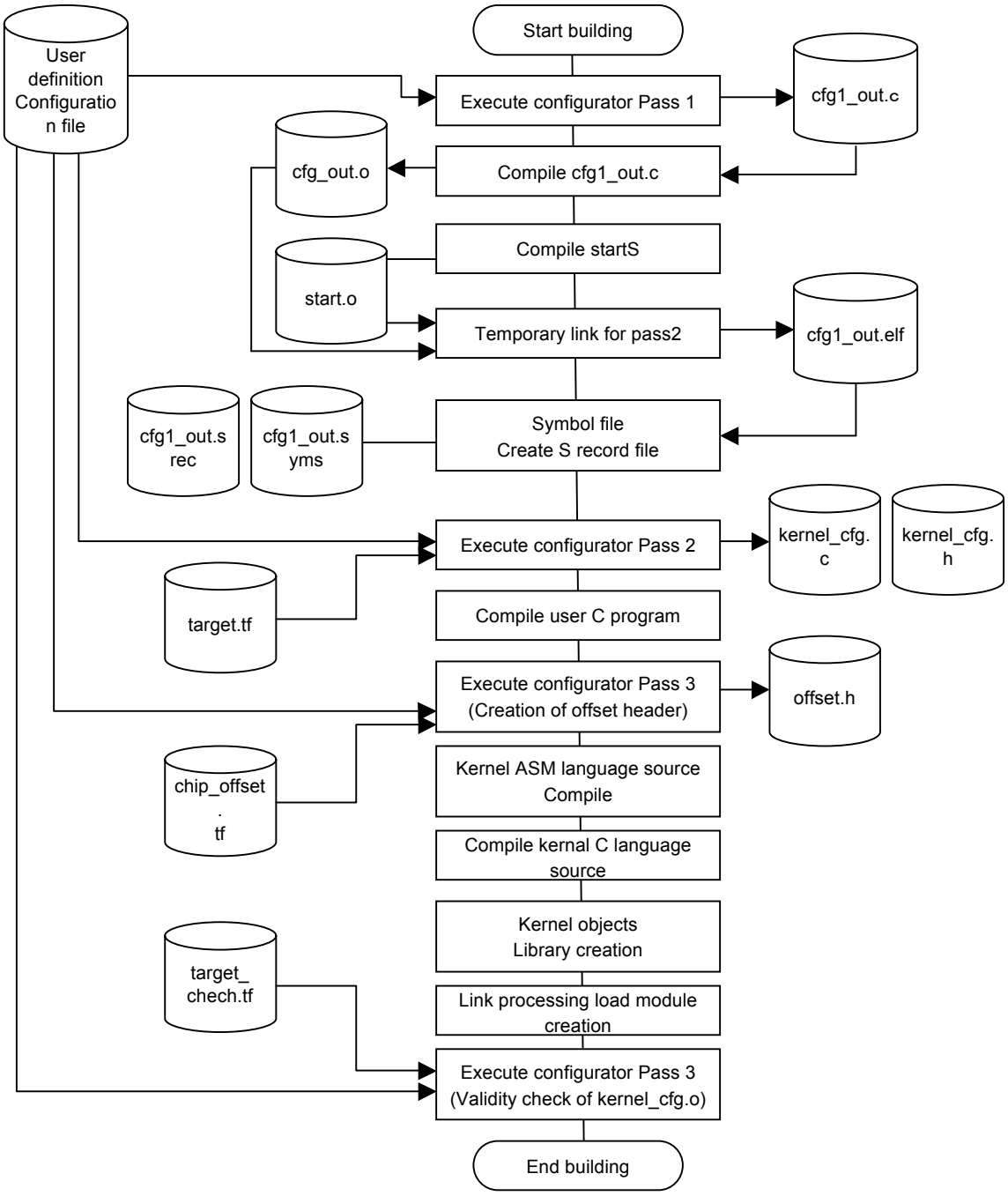

Figure 3.3 Execution timing of build process and configurator

Execution process of configurator

It is necessary to activate the configurator three times to build a system on New Generation Kernel. Each of the activations of the configuration is called "Pass 1", "Pass 2" and "Pass 3" respectively from the first activation. The number that specifies a pass like '1' in Pass 1 is called "Pass Number". The Pass Number is specified with a configurator argument --pass<pass number> option.

In each pass, a file is created if necessary, which may be used by a subsequent pass or for

compiling and link.

The behaviour of a configurator in response to <Pass Number> is described hereunder.

Pass 1:
The following configurator pre-process is provided in Pass 1:
In concrete, the system configuration file is interpreted and create C-language source file called 'cfg_out.c'.
'cfg.out.c' is filled with the parameters of the static API file described in the system configuration file, the equation to check the characteristics of C-language processing system to be used, the type information dependent on the kernel and software part and the parameters defined by the application.
'cfg_out.c' compiles using C Compiler and linker, establishes a temporary link and generates an S record and a symbol table. At this point, it is necessary to name the S record file 'cfg_out.srec' and the symbol table 'cfg_out.syms'. By doing so, when using --cfg_out option, the file name specified will be applied instead of the default file name.

Pass 2:
The template file that is interpreted and executed in Pass 2 contains a template that creates two files, namely 'kernel_cfg.h' and 'kernel_cfg.c'. By embedding the information with which the configurator provided the macro-processor, a source file to define the kernel configuration and the initial state is created.
The template file generates the kernel object initialisation table, the management table area, the stack area and the dispatch routine to call interrupt service routine.
It also generates vector code of processor if applicable.
The segment that outputs variables, constants, codes and the stack area is output to an implicit segment of the processing system by default.
In case controlling of the variable area allocation that is generated by the script of the linker to be used is desired, it is possible to add a segment name that can be identified by the linker's script using the extended language function in C-language scheme. In the memory architecture of a multi-core system, local memory and private memory are assumed.
To cope with them, it is possible to control the allocation of the variable area by customising the template file.

Pass 3:
Pass 3 makes the offset information and mask information output to offset.h for accessing the C-language defined structure members and bit field members from ASM language. This process is required to be executed before compiling the kernel ASM source.
The template file provided defines the template to be used for output to offset.h.

*This was a function provided as perl script genoffset in the kernels before New Generation (like TOPPERS/JSP).

In Pass 3, it is also possible to execute the checker function to decide whether the kernel is configured normally after the completion of the total system links.
The parameter check process of a static API, an error of which could not be detected in Pass 2, is described in the template file to be executed. Also, the compatibility to the memory

border dependent on the target hardware can be checked.
Even if Pass 3 of the checker is omitted, there is no change in the content to be built. However, user is responsible for detecting errors that should have been detected originally by the configurator.
* This is a function that was provided by a command (chk) different from the configurator (cfg) in the kernels before New Generation (like TOPPERS/JSP).

Pass 4:
Pass 4 is not executed in TOPPERS/ASP. It is provided only for the kernels (HRP, HRP2) that have the memory protection function.

3.7.2. Types of static API parameters

As a result of reviewing the processing method of the configuration, the classification of the static API parameters defined in μITRON Ver 4.0 Specification has also been reviewed in New Generation Kernel Specification. The classification of the parameters of static APIs in New Generation Kernel Specification is described hereunder.

① Object ID
A unique name that identifies an object. The configurator automatically allocates ID Number in response to this and outputs to kernel_cfg.h as a C constant macro.
In μITRON 4.0 Specification, it was possible to specify user-assigned ID number direct value or ID number in C-language macro and they were classified as "auto-assignable integer value parameter" In New Generation Kernel, ID of an object can only be auto-assigned by the configurator. Neither user-assigned ID number immediate value nor ID number in C-language macro can be specified as an argument of the object ID of a static API.

② Integer expression parameter
The parameter that specifies an integer value including the object number, function code, object attribute, size, number of pieces, priority, etc. It can be specified with C-macro deployed by C pre-processor.
It is possible to describe an integer constant expression in which the value is decided independent of the address where the program is allocated.
This type includes the interrupt handler number not subject to auto-assignment and CPU exception handler number.

③ General constant express parameter
Parameters that may specify an address including an entry address per process basis, the starting address of the memory area and extension information are classified to this group. A given constant expression can be described. It can be specified with C-macro deployed by C pre-processor.

④ String parameter
Parameters that specify a string such as an object module name and a section name.
Any given string can be described according to the description method in C-language.
There is no static API in TOPPERS/ASP that has a string parameter as an argument.

4. STATIC API REFERENCE

This chapter describes the kernel static APIs.

4.1. Inclusion of INCLUDE configuration file

[Explanation]
A configuration file defined externally is incorporated.

[Declaration]
```
INCLUDE( ”cfgfile” );
```

[Parameter]
cfgfile : An external configuration file path is specified.

[Relation with µITRON 4.0 Specification]
In µITRON Specification, INCLUDE static API was a static API in which INCLUDE of the header file defined in the kernel configuration and the initialisation file (kernel_cfg.c) was created and specified in the pre-processor directive format in C-language (#include ･･･) .
In New General Kernel Specification, the identical directive (#iinclude) to the include directive (#include) of the C-language processor included in the system configuration file shall be created in the kernel configuration and initialisation file (kernel_cfg.c)
The order of the directives to be created is consistent with the order of the include directives in the system configuration file.
On the other hand, INCLUDE static API is limited to the use of incorporating the configuration file (*.cfg) for comprising external definition. With this specification modification, only static APIs can be described in the file defined with INCLUDE static API.

4.2. CRE_TSK Create Task

[Explanation]
It defines task creation.

[Definition method]
```
CRE_TSK( ID tskid ,{ ATR tskatr、 intptr_t exinf, TASK task,PRI
itskpri,SIZE stksz,STK_T *stk } );
```

[Parameter]

ID tskid	:	Identification of task
ATR tskatr	:	Task attribute
intptr_t exinf	:	Extension information of Task
TASK task	:	Task activation address
PRI itskpri	:	Priority at task activation
SIZE stksz	:	Stack area size of task
STK_T *stk	:	First address of stack area of task

Description of Parameters :

ID tskid : Identification of task
ID number is auto-assigned by the configurator.
The auto-assigned ID number is output to kernel_cfg.h with the specified identification as C constant macro.

ATR tskatr : Task attribute
In case TA_ACT is specified The task shall be executable at the same time as its creation.
In case TA_ACT is not specified, the task becomes dormant at the same time as its creation.
In that case, TA_NULL shall be specified.

intptr_t exinf : Extension information of Task
It is information passed as an argument to the main function of a task or a task exception.

TASK task : Task activation address
The task activation address shall be specified. In general, it is defined with the entry function of a task.

PRI itskpri : Priority at task activation
It can be specified with the range from 1 to 16.
1 represents the highest priority while 16 the lowest.

SIZE stksz : Stack area size of task
The size of the stack area that a task uses is specified by bytes.

STK_T *stk : First address of stack area of task

For other than specifying NULL, the starting address of the stack area reserved by user shall be defined.

In general a variable of C shall be specified.

In case NULL is specified, the stack area of a task is reserved by the size configuration specified.

[Description format of task in C-language]

```
void task( intptr_t p_exinf )
{
       <Description of process of task itself>
       ext_tsk(); /* Task exit (omissible)*/
}
```

The last ext_tsk() is omissible and, if omitted, ext_tsk() is executed when the main function of the task returns and the system behaves the same.

4.3. DEF_TEX　Define Task Exception Handling Routine

[Explanation]
The task exception handling routine of the task to be created is defined.

[Definition method]
```
DEF_TEX(ID tskid, {ATR texatr,TEXRTN texrtn } );
```

[Parameter]
ID tskid　　:　Object ID of task to be defined
ATR texatr　　:　Attribute of task exception handling routine
TEXRTN texrtn　:　Activation address of task exception handling routine

Description of Parameters :
ID tskid　　:　Object ID of task to be defined
The identifier of the task in which a task exception is defined shall be described.
It is necessary to define the declaration of the target task before this declaration.

ATR texatr　　:　Attribute of task exception handling routine
Only TA_NULL can be specified.

TEXRTN texrtn　:　Activation address of task exception handling routine
The activation address of a task exception handling routine shall be specified. In general C function name is specified.

[Description format of task exception process in C-language]

```
void task_tex_routine( TEXPTN texptn, intptr_t p_exinf )
{
       <Description of task exception handling>
}
```

TEXPTN texptn:　:　Task exception factor occurred (32 bit)
intptr_t　p_exinf　:　Extension information of task

4.4. CRE_SEM　Create Semaphore

[Explanation]
The semaphore to be created shall be defined.

[Definition method]

```
CRE_SEM( ID semid, { ATR sematr, UINT isemcnt,UINT maxsem } );
```

[Parameter]

ID semid	:	Object ID of semaphore
ATR sematr	:	Attribute of semaphore
UINT isemcnt	:	Initial value of semaphore resource
UINT maxsem	:	Maximum number of resources of semaphore

Description of Parameters :

ID semid　　:　Object ID of semaphore

ID number is auto-assigned by the configurator.
The auto-assigned ID number is output to kernel.cfg.h with the specified identification as C constant macro.

ATR sematr　　:　Attribute of semaphore

TA_TPRI can be specified as an attribute specifiable.
In case TA_TRPI is specified, the queue of the semaphore shall be in the order of the task priority.
In case TA_TRPI is not specified (TAL_NULL), then the default FIFO shall be applied.
TA_TFIFO, which is FIFO attributable, was abolished in New Generation Kernel, however, it is left to maintain the compatibility with μITRON specification. Instead of TA_NULL, TA_TFIFO may be specified.

UINT isemcnt　　:　Initial value of semaphore resource

The initial value of the number of the semaphore resource is specified. The initial value of the number of the resources can be specified in the range from 0 to ～0xffffffff.

UINT maxsem　　:　Maximum value of the semaphore resources

The maximum value of the semaphore resources is specified. The maximum value of the number of the semaphore resource can be specified in the range from 0 to 0xffffffff.

4.5. CRE_FLG Create Event Flag

[Explanation]
The event flag to be created is defined.

[Definition method]
```
CRE_FLG( ID flgid, { ATR flgatr,FLGPTN  iflgptn } );
```

[Parameter]
ID flgid : Object ID of the event flag
ATR flgatr : Attribute of the event flag
FLGPTN iflgptn : Initial value of bit pattern of the event flag

Description of Parameters :
ID flgid : Object ID of the event flag
ID number is auto-assigned by the configurator.
The auto-assigned ID number is output to kernel_cfg.h with the specified identification as C constant macro.

ATR flgatr : Attribute of the event flag
In case TA_CLR is specified, all the bit patterns of the event flag are all clear to 0 when the task is released from the waiting state. The default is non-0 clear attribute.
An event flag with TA_CLR not specified is not cleared to zero when released from waiting.
By default, more than one task cannot wait at the same time for the target event flag.
For an event flag created expressly with TA_WMUL, more than one tasks can wait as target.
In case of TA_WMUL attribute, TA_TPRI (scheme following the order of the priority) can be specified as the task waiting method. In case TA\TPRI is not specified, the waiting method will be FIFO (by default).
TA_WSGL and TA_FIFO that are the default attributes (value 0) are defined in itron.h for the compatibility with µITRON Specification.

FLGPTN iflgptn : Bit pattern initial value of event flag
The initial value of the bit pattern at the creation of the event flag is specified. In ASP, 32-bit value can be specified as the number of bits of the event flag.

4.6. CRE_DTQ Create Data Queue

[Explanation]
The data queue to be created is defined.

[Definition method]
```
CRE_DTQ( ID dtqid, { ATR dtqatr, uint_t  dtqcnt, void *dtqmb } );
```

[Parameter]
ID dtqid : Object ID of data queue
ATR dtqatr : Attribute of data queue
uint_t dtqcnt : Capacity of data queue area (the number of data)
void *dtqmb : The starting address of data queue management area

Description of Parameters :
ID dtqid : Object ID of data queue
ID number is auto-assigned by the configurator.
The auto-assigned ID number is output to kernel_cfg.h with the specified identification as C constant macro.

ATR dtqatr : Attribute of data queue
The send queue method of task is FIFO by default.
In case TA_TPRI attribute is specified, the send queue method will the order of the task priority.
The send queue of data queue is always FIFO as in ITRON Specification.
If created with FIFO attribute, TA_NULL can be specified.
*TA_TFIFO of the default attribute (calue 0) is defined in itron.h for the compatibility with µITRON Specification.

uint_t dtqcnt : Capacity of data queue area (the number of data)
In dtqcnt, the number of data is specified as the data queue capacity.
1 data size of a data queue shall be 1 word (32-bit integer).
0 can be specified as the number of data. A data queue with the number of data being 0 can be used as a synchronisation object that notifies one UNIT-type data.

void *dtqmb : Starting address of data queue management area
Only NULL can be specified. The necessary size of the data queue area is reserved by the configurator.

4.7. CRE_PDQ Create Priority Data Queue

[Explanation]

It creates a priority data queue.

A priority data queue is an object newly introduced in "TOPPERS New Generation Kernel Integration Specification". It does not exist in μITRON Specification.

A priority data queue is an object that communicates one-word message like data queue. The difference is that the message data has the priority.

[Definition method]

```
CRE_PDQ( ID pdqid, { ATR pdqatr, uint_t pdqcnt, PRI maxdpri,void
*pdqmb } );
```

[Parameter]

ID pdqid	:	Object ID of priority data queue
ATR pdqatr	:	Attribute of priority data queue
uint_t pdqcnt	:	Capacity of priority data queue area
PRI maxdpri	:	Highest data priority
void *pdqmb	:	Management area of priority data queue

Description of Parameters :

ID pdqid : Object ID of priority data queue

ID number is auto-assigned by the configurator.

The auto-assigned ID number is output to kernel_cfg.h with the specified identification as C constant macro.

ATR pdqatr : Attribute of priority data queue

As an attribute of the send queue, the task priority order TA_TPRI can be specified. FIFO order is specified by default.

If created with FIFO order attribute, TA_NULL can be specified.

*TA_TFIFO of the default attribute (value 0) is defined in itron.h for the compatibility with μITRON Specification.

*The receive queue of priority data queue is always FIFO.

uint_t pdqcnt : Capacity of priority data queue area

The number of data is specified as the priority data queue capacity in pdqcnt.

The size of 1 data of priority data queue is 1 word (32 bit integer).

0 can be specified as the number of data. The priority data queue with the number of data being 0 can be used as a synchronisation object that notifies a data of unit_t type.

PRI maxdpri : Highest data priority

The maximum value of the data priority that can be sent to the priority data queue created.

void *pdqmb : Management area of priority data queue
Only NULL can be specified. The configurator reserves the necessary size of the management area of a priority data queue.

[Difference from µITRON 4.0 Specification]
It is an object that does not exist in µITRON 4.0 Specification.

4.8. CRE_MBX Create Mailbox

[Explanation]
It defines a mailbox to be created.

[Definition method]
```
CRE_MBX( ID mbxid, { ATR mbxatr, PRI maxmpri, void * mbxmb } );
```

[Parameter]
ID mbxid : Object ID of mailbox
ATR mbxatr : Attribute of mailbox
PRI maxmpri : Highest priority of mailbox
void *mbxmb : Starting address of mailbox management area

Description of Parameters :
ID mbxid : Object ID of mailbox
ID number is auto-assigned by the configurator.
The auto-assigned ID number is output to kernel_cfg.h with the specified identification as C constant macro.

ATR mbxatr : Attribute of mailbox
As specifiable attribute, TA_TPRI and TA_MPRI can be specified.
The receive queue of mailbox is FIFO order by default, but if TA_TPRI is specified, it will follow the task priority order.
The message queuing order is also FIFO order by default. If TA_MPRI is specified, it will follow the message priority.
If created with FIFO order attribute, TA_NULL can be specified.
*TA_TFIFO and TA_MFIFO with the default attribute (value 0) are defined in itron.h for the compatibility with µITRON Specification.

PRI maxmpri : Highest priority of mailbox
In case the attribute of the mailbox created is TA_MRPI. the maximum value of the message priority shall be set. For the message priority can be specified with the range from 1 to max. 16.
In case of TA_MFIFO attribute, this parameter is ignored.

void *mbxmb : Starting address of mailbox management area
Only NULL can be specified. The configurator reserves the area necessary for management of the maximum number of message queueing.

4.9. CRE_MFP Create Fixed-length Memory Pool

[Explanation]
It defines the fixed-length memory pool to be created.

[Definition method]
```
CRE_MPF(ID mpfid, { ATR mpfatr, uint_t blkcnt, uint_t blksz,MPF_T *mpf,
void *mpfmb } );
```

[Parameter]
ID mpfid : Object ID of fixed-length memory pool
ATR mpfatr : Attribute of fixed-length memory pool
uint_t blkcnt : Number of blocks of fixed-length memory pool
uint_t blksz : 1Size of 1 block (bytes)
MPF_T *mpf : Starting address of fixed-length memory pool management area
void *mpfmb : Starting address

Description of Parameters :
ID mpfid : Object ID of fixed-length memory pool
ID number is auto-assigned by the configurator.
The auto-assigned ID number is output to kernel_cfg.h with the specified identification as C constant macro.

ATR mpfatr : Attribute of fixed-length memory pool
TA_TPRI can be specified as an attribute specifiable.
The queue waiting for acquiring of fixed-length memory pool is FIFO order by default. If TA_TRPI is specified, it will follow the task priority. If created with FIFO order attribute, TA_NULL can be specified.
*TA_TFIFO of the default attribute (value 0) is defined in itron.h for the compatibility with μITRON Specification.

uint_t blkcnt : Number of blocks of fixed-length memory pool
The number of the blocks of the fixed-length memory pool to be created is specified.

uint_t blksz : 1Size of 1 block (bytes)
The size of 1 block of the fixed-length memory pool to be created is specified in bytes.

MPF_T *mpf : Starting address of fixed-length memory pool management area
Only NULL can be specified. The configurator reserves the necessary size for the fixed-length memory pool area.

void *mpfmb : Starting address
Only NULL can be specified.

The configurator reserves the size required.

4.10. CRE_CYC　Create Cyclic Handler

[Explanation]
It creates a cyclic handler.
If TA_STA is specified in cycatr, the specified cyclic handler will be put into operation at the time of creation.
The time when a cyclic handler is activated is set to the specified relative time from the time when the service call (sta_cyc) is called (the time of activation of the kernel in case of a static API). cycphs can be set to a larger value than that of cyctim.
If TA_STA is specified with cycatr and cycphs with 0, the time when the cyclic handler is called for the first time is the first time tick after the initial activation of the kernel.
Also if cycphr is specified with 1, the system behaves in the same way. Therefore, if cycatr is specified with TA_SYA for a static API, it is not recommended to specify cycphs with 0. If so specified, the configurator raises a warning message.

[Definition method]

```
CRE_CYC( ID cycid, { ATR cycatr, intptr_t exinf, CYCHDR cychdr, RELTIM
cyctim, RELTIM cycphs } );
```

[Parameter]

ID cycid	:	Object ID of cyclic handler
ATR cycatr	:	Attribute of cyclic handler
intptr_t exinf	:	Extension information of cyclic handler
CYCHDR cychdr	:	Activation address of cyclic handler
RELTIM cyctim	:	Activation cycle of cyclic handler
RELTIM cycphs	:	Activation phase of cyclic handler

Description of Parameters :

ID cycid　: Object ID of cyclic handler
ID number is auto-assigned by the configurator.
The auto-assigned ID number is output to kernel.cfg.h with the specified identification as C constant macro.

ATR cycatr　: Attribute of cyclic handler
If TA_STA attribute is specified, the cyclic handler is activated at the same time of creation.
If not, the cyclic handler is stopped at the time of creation.
* ASP does not support TA_PHS (phase storing) attribute.

intptr_t exinf　: Extension information of cyclic handler
It specifies the extension information of cyclic handler. This parameter is passed as an argument for activation of the cyclic handler.

CYCHDR cychdr　: Activation address of cyclic handler

It specifies the activation address of cyclic handler.

RELTIM cyctim : Activation cycle of cyclic handler
It specifies the activation phase of cyclic handler in millisecond

RELTIM cycphs : Activation phase of cyclic handler
The time of activation of a cyclic handler is specified in cycphs as the relative time from the time when the activation service call is called (the time of activation of the kernel in case a static API is specified with TA_STA attribute).
In μITRON Specification, it is not recommended to set cycphs with a larger value than that of cyctim. If such setting is done, it provided that the behaviour shall be defined by the implementation.
In New Generation Kernel Specification, as the activation time is changed from ctctim to cycphs, cycphs can be set with a larger value than that of cyctim.

[Description format of cyclic handler in C-language]

```
void cychdr( intptr_t exinf )
{
<Description of cycle hander process]
}
```

[Difference from μITRON 4.0 Specification]
cycdhr type is changed to CTCHDR.
The activation time is specified with cycphs instead of cyctim.
Specifying cycphs with a larger value than that of cyctim is allowed and that behaviour is specified.
The behaviour at the time when cycphs is specified with 0 for a static API is specified.

4.11. CRE_ALM　Create Alarm Handler

[Explanation]
It creates an alarm handler
An alarm handler is created with the initial state being inactive.

[Definition method]
```
CRE_ALM( ID almid, { ATR almatr, intptr_t exinf, ALMHDR almhdr } );
```

[Parameter]
ID almid　　:　Object ID of alarm handler
ATR almatr　　:　Attribute of alarm handler
intptr_t exinf　　:　Extension information of alarm handler
ALMHDR almhdr　:　Activation address of alarm handler

Description of Parameter :
ID almid　　:　Object ID of alarm handler
ID number is auto-assigned by the configurator.
The auto-assigned ID number is output to kernel.cfg.h with the specified identification as C constant macro.

ATR almatr　　:　Attribute of alarm handler
Only TA_NULL can be specified.

intptr_t exinf　　:　Extension information of alarm handler
It specifies the extension information of alarm handler. This parameter is passed as an argument for activation of the alarm handler.

ALMHDR almhdr　:　Activation address of alarm handler
It specifies the activation address of an alarm handler

[Description format of alarm handler in C-language]

```
void cychdr( almhdr(intptr_t exinf )
{
<Description of alarm handler>
}
```

[Difference from μITRON 4.0 Specification]
almhdr type is changed to ALMDR.

4.12. CFG_INT　Configure Interrupt Request Line Attribute

[Explanation]

To an interrupt request line specified with intno (target interrupt request line), the attribute and the interrupt line priority specified with the relevant parameters are configured.

The number that identifies an interrupt request line is the interrupt number.

It is necessary to define an interrupt number in the range specified by the CPU dependent part from TMIN_INTNO to TMAX_INTNO.

An interrupt number is determined according to the specification of the target hardware and defined in the CPU dependent part based on natural numbering. Therefore, the number is not necessarily a positive value.

The number of levels for the interrupt priority is dependent on CPU.

A larger value than TMIN_INTPRI and less than TMAX_INTRPI , which are defined by the CPU dependent part, can be specified.

In case TA_ENAINT is specified as line attribute, the initial state of the interrupt line state is set with the interrupt-enabled state.

The following definitions are available dependent on the CPU and the interrupt controller as attributes.

TA_POSEDGE The specified line is set as a positive edge trigger.
TA_NEGEDGE The specified line is set as a negative edge trigger.
TA_BOTHEDGE The specified line is set as a both-edge trigger
TA_LOWLEVEL The specified line is set as a low level trigger.
TA_HIGHLEVEL The specified line is set as a high level trigger.

See the manual of the dependent part which attribute can be defined.

[Definition method]

```
CFG_INT( INTNO intno, { ATR intatr, PRI intpri } );
```

[Parameter]

INTNO intno : Interrupt number
ATR intatr : Interrupt request line attribute
PRI intpri : Interrupt priority

*intno, intatr and intpri are integer constant expression parameters.

Description of Parameter:

INTNO intno : Interrupt number

It specifies the interrupt number that specifies the interrupt line in which the interrupt attribute is configured.

The interrupt number is target-dependent definition.

ATR intatr : Interrupt request line attribute

It specifies the interrupt trigger configuration as an attribute. The following macros are

available:

TA_POSEDGE The specified line is specified as a positive edge trigger.
TA_NEGEDGE The specified line is set as a negative edge trigger.
TA_BOTHEDGE The specified line is set as both edge triggers.
TA_LOWLEVEL The specified line is set as a low level trigger.
TA_HIGHLEVEL The specified line is set as a high level trigger.

It depends on the target which macro can be used.

PRI intpri : Interrupt priority

It specifies the priority of an interrupt that occurs. For the interrupt priority, a consecutive negative number starting from -1 is used.
The less the value is (the larger the absolute value is), the higher the priority is.
This is the common concept of New Generation Integration Kernel Specifications different from the interrupt priority configured to hardware. The level of the priority is dependent on the target.

[Difference from µITRON 4.0 Specification]
It is a newly introduced static API not specified in µITRON 4.0 Specification.

4.13. DEF_INH　Define Interrupt Handler

[Explanation]
It defines the interrupt handler and its attribute value to the interrupt handler number.
The interrupt handler number shall be defined in the range from YMIN_INHNO to TMAX_INHNO defined by the CPU dependent part.
When the interrupt handler prepared by user is registered in the kernel, the number that identifies the interrupt to be registered by the interrupt handler is called an interrupt handler number
By nature, an interrupt handler number shall correspond 1 to 1 with an interrupt number, in principle.　In many cases, both have the same value. (but not mandatory to have the same value)
As with the interrupt number (INTNO), the interrupt handler number (INHNO) shall be numbered in a natural numbering form determined by the target hardware specification. Therefore, the number is not necessarily a positive value.
The interrupt request line attribute that corresponds with the target interrupt handler number must be configured with CGF_INT static API.
To the corresponding interrupt line, any other interrupt handler or interrupt service routine must not be registered in duplicate.

[Definition method]
```
DEF_INH ( INHNO inhno, { ATR inhatr, INTHDR inthdr } );
```

[Parameter]
INHNO inhno　:　Interrupt handler number to be defined
ATR inhatr　:　Interrupt handler attribute
INTHDR inthdr　:　Activation address of interrupt handler
* inhno and inhatr are integer constant express parameters and inthdr is a general constant parameter.

Description of Parameters:
INHNO inhno　:　Interrupt handler number to be defined

ATR inhatr　:　Interrupt handler attribute
In case any interrupt handler not under control of the kernel is supported as an attribute, TA_NONKERNEL attribute may be specifiable. It is dependent on the target whether TA_NONKERNEL attribute can be specified or not.

INTHDR inthdr　:　Activation address of interrupt handler
It specifies the activation address of an interrupt handler.

[Description format of interrupt handler in C-language]

```
void inthdr( void )
{
       <Description of interrupt handler process>
}
```

[Difference from µITRON 4.0 Specification]
inthdr data type is changed to INTHDR.

4.14. DEF_EXC　Define CPU Exception Handler

[Explanation]
CPU exception handler is defined to the CPU exception handler number specified by excno. User can register a CPU exception handler for each type of CPU exception that the processor detects.
The number that identifies a CPU exception to be registered by a CPU exception handler is called CPU exception handler number. A CPU exception handler number is expressed in EXCNO type and numbered basically according to natural numbering determined by the target hardware specification. Therefore, the number is not necessarily a positive value.

[Definition method]
```
DEF_EXC( EXCNO excno, { ATR excatr, EXCHDR exchdr } );
```

[Parameter]
EXCNO excno　:　CPU exception handler number to be defined
ATR excatr　:　CPU exception handler attribute
EXCHDR exchdr　:　Activation address of CPU exception handler
* excno and excatr are integer constant expression parameters and exchdr is a general constant expression parameter.

Description of Parameters:
EXCNO excno　:　CPU exception handler number to be defined
The number that specifies a CPU expression handler. CPU exception handler number is dependent on the target.

ATR excatr　:　CPU exception handler attribute
Only TA_NULL can be specified.

EXCHDR exchdr　:　Activation address of CPU exception handler
It specifies the activation address of a CPU exception handler.

[Description format of CPU exception handler in C-language]
```
void exchdr( VP p_exc_inf )
{
       <Description of CPU exception handler>
}
```

[Difference from µITRON 4.0 Specification]
None.

4.15. DEF_ICS　Set Stack Area for Non-Task Context

[Explanation]
It creates a stack area for non-task context according to the configuration information of the stack area for non-task context specified by the parameter.
The stack area for non-task context is a stack area independent of the stack area of the task used mainly by the interrupt process. If multiple interrupts are supported, it is necessary to reserve sufficient size for this area.
Unless user configures the stack area for non-task context using DEF_ICS, the default-sized stack area defined by the target is reserved by the configurator.

[Definition method]
```
DEF_ICS( { SIZE istksz, STK_T *istk } );
```

[Parameter]
SIZE istksz　　: Size of stack area for non-task context (bytes)
STK_T *istk　　: Starting address of the stack area for non-task context

Description of Parameters:
SIZE istksz　　: Size of stack area for non-task context (bytes)
istkzs is an integer constant express parameter and istk is a general constant expression parameter.
istksz cannot be specified with 0, or with a value smaller than the minimum value defined by the target.

STK_T *istk　　: Starting address of the stack area for non-task context
If istksz is NULL, the stack area of the size specified by istksz is reserved by the configurator. In case a size that does not comply with the restriction of the definition by the target is set to istksz, the size rounded up to a larger value will be reserved so that the restriction of the definition by the target is met.
If any other value than NULL is specified to istk, it is necessary for user to specify the area reserved by user for the stack area specified by istk and istksz.

[Difference from µITRON 4.0 Specification]
It is a newly introduced static API not defined in µITRON 4.0 Specification.

4.16. ATT_ISR　Add Interrupt Service Routine

[Explanation]
It registeres an interrupt service routine by specifying an interrupt line.
An interrupt service routine created by ATT_ISR does not have ID number.
It is possible to register more than one interrupt service routine in an interrupt line.
If multiple service routines are registered in an interrupt line, the order of the prioritized process is decided by the interrupt service routine priority.
The attribute of the corresponding interrupt request line shall be configured with CGF_INT static API.
Any other interrupt handler must not be specified to the corresponding interrupt line in duplicate.
Also, it is not possible to specify an interrupt not managed by the kernel using intno.

[Definition method]
```
ATT_ISR( { ATR isratr, intptr_t exinf, INTNO intno, ISR isr, PRI isrpri } );
```

[Parameter]
ATR isratr　　:　Interrupt service routine attribute
intptr_t exinf　　:　Extension information of interrupt service routine
INTNO intno　　:　Interrupt number to which interrupt service routine is registered
ISR isr　　:　Starting address of interrupt service routine
PRI isrpri　　:　Interrupt service routine priority
* isratr, intno and isrpri are integer constant expression parameters and exinf and isr are general constant expression parameters.

Description of Parameters:
ATR isratr　　:　Interrupt service routine attribute
It specifies the attribute of an interrupt service routine. Only TA_NULL can be specified.

intptr_t exinf　　:　Extension information of interrupt service routine
It specifies the extension information that is passed as an argument to an interrupt service routine.

INTNO intno　　:　Interrupt number to which interrupt service routine is registered
The interrupt line to which the interrupt service routine is added is specified with an interrupt number (intno).

ISR isr　　:　Starting address of interrupt service routine
It specifies the starting address of an interrupt service routine.

PRI isrpri　　:　Interrupt service routine priority
It specifies the priority of an interrupt service routine. The interrupt service routines added to

the same interrupt line are called in the order of the priority specified.
The priority of the interrupt service routine is specified with a consecutive positive number from 1 and 1 represents the highest priority. For ASP, the number of the levels are fixed to 16 and this represents the lowest priority. The maximum value of the priority is set to TMAX_ISRPRI.

[Description format of interrupt service routine in C-language]

```
void isr( intptr_t exinf )
{
        <Description of interrupt service routine process>
}
```

[Difference from µITRON 4.0 Specification]
isrpri (interrupt service routine interrupt priority), which determines the processing order has been added to the creation information of an interrupt service routine.

4.17. ATT_INI　Add Initialised Routine

[Explanation]

It adds the initialization process of user called at the kernel activation.

This routine is called prior to the execution of the task and the interrupt handler.

Execution of the initialisation process, etc. prior to the execution of a user application is assumed.

As it is called from an execution context of the kernel, many service calls cannot be called.

The order of calling from the kernel shall be the order defined in this API.

[Definition method]

ATT_INI({ ATR iniatr, intptr_t exinf, INIRTN inirtn });

[Parameter]

ATR iniatr　　:　Initialisation routine attribute

intptr_t exinf　　:　　Initialisation routine extension information

INIRTN inirtn　　:　　Activation address of initialisation routine

Description of Parameters:

ATR iniatr　　:　　Initialisation routine attribute

Only TA_NULL can be specified.

intptr_t exinf　　:　Initialisation routine extension information

It is information that is passed as an argument to an initialisation routine.

INIRTN inirtn　　:　Activation address of initialisation routine

It specifies the activation address of an initialisation routine.

[Description format of initialisation routine in C-language]

```
void inirtn( intptr_t exinf )
{
       <Description of initialisation routine process>
}
```

[Difference from μITRON 4.0 Specification]

In TOPPERS New Kernel Integration Specification, the type of the extension information and the activation address has been changed.

4.18. ATT_TER　Add Termination Process Routine

[Explanation]
It adds a termination process routine that is called at the kernel activation.
As it is called from an execution context of the kernel, many service calls cannot be called.
A termination routine that is executed in a specific core shall be defined in a class enclosure.
The order of calling by the kernel shall be order according to the definition of this API.

[Definition method]
ATT_TER({ ATR teratr, intptr_t exinf, TERRTN terrtn });

[Parameter]
ATR teratr　　:　Termination process routine attribute
intptr_t exinf　　:　Termination process routine extension information
TERRTN terrtn　　:　Starting address of termination process routine
* teratr is an integer constant expression parameter and exint and terrtn are general constant expression parameters.

Description of Parameters:
ATR teratr　　:　Termination process routine attribute
Only TA_NULL can be specified.

intptr_t exinf　　:　Termination process routine extension information
This information is passed as an argument to a termination process routine.

TERRTN terrtn　　:　Starting address of termination process routine
It specifies the activation address of a termination process routine.

[Description format of termination routine in C-language]

```
void terrtn( intptr_t exinf )
{
       <Description of termination routine process>
}
```

[Difference from µITRON 4.0 Specification]
It is a newly introduced static API not defined in µITRON 4.0 Specification.

5. KERNEL DYNAMIC APIs (SERVICE CALL) REFERENCES

This chapter describes kernel dynamic APIs (service calls)

5.1. Header File

Application developers that use the kernel APIs shall include kernel.h.
itron.h is also contained for the compatibility with μITRON Ver 4.0.
In case of using the data types and the constants that were specified in μITRON Specification but have been abolished in "TOPPERS New Generation Kernel Integration Specification", the ITRON Specification compatibility file (itron.h) shall be included.

5.2. Task Management Function

5.2.1. act_tsk/iact_tsk Activate Task

[Function]
Activate the task specified by tskid.
When the target task is dormant, the initialisation process that should be done at the same time with the task activation is performed to the target task, and the target task becomes the executable state.
When the target task is not dormant, 1 is added to the activation request queue of the target task. In case the number of the activation request queue exceeds TMAX_ACTCNT, it results in E_QOVR error.
In ASP, TMAX_ACTNCT is fixed to 1. The number of pending activation requests is 1.
When the task is terminated while the activation request is queuing, it becomes immediately the state executable.
If TSK_SELF (= 0) is specified in tskid in act_tsk, the own task becomes the target task.

[Format]
Call from task part : ER ercd = act_tsk (ID tskid);
Call from non-task part : ER ercd = iact_tsk(ID tskd);

[Parameter]
ID tskid : ID number of task to be activated

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
Called while CPU is locked
act_tsk was called from non-task part
iact_tsk was called from task part.
E_ID : Incorrect ID number (tskid is incorrect or cannot be used)
E_QOVR : Queuing overflow (overflow of activation request queues)

[Definition of implementation dependency]
The number of activation request queues is 1, meaning that the activation request is held once.

5.2.2. can_act　Cancel task activation request

[Function]
All the activation requests of the target tasks are cancelled.
As the return value, the number of the activation requests cancelled is returned.
In concrete, the number of the activation requests in queue of the target task is set to 0, the number of the activation requests in queue before it is set to 0 is returned as the return value of the service call
If TSK_SELF (=0) is specified in tskid, the own task becomes the target task.

[Format]
ER_UINT actcnt = can_act(ID tskid);

[Parameter]
ID tskid　　:　Task ID number whose task activation request is to be cancelled.

[Return value]
ER_UINT actcnt　:　Times of an activation request that has been queued (0 or more), or an error code.

Value equal to 0 or more	:	Times of an activation request that has been queued.
E_ID	:	Incorrect ID number (tskid is incorrect or cannot be used)
E_CTX	:	Context error Called while CPU is locked Called from non-task part.

[Definition of implementation dependency]
None.

5.2.3. ext_tsk　Terminate Own Task

[Function]
It terminates the own task.
In concrete, the process that should be done at its termination is performed to the own task and the own task becomes dormant state. Then, in case the number of the activation requests in queue is not 0, the process that should be done at its activation is performed to the own task and the own task becomes executable.
At this time, the number of the activation requests in queue is decremented by 1.
If ext_tsk is normally processed, ext_tsk returns no value.
In principle, ext_tsk is called in CPU unlocked state, all interrupt priority masks released state and dispatched permitted state, but if it is called in any other state, the state is transitioned to CPU unlocked state, all interrupt priority masks released state or dispatched permitted state and then the own task is terminated.

[Format]
ER ercd = ext_tsk(void);

[Parameter]
None.

[Return value]
If ext_tsk is normally processed, ext_tsk returns no value.
The following error codes are returned in case of error.

ER ercd	:	Error code
E_SYS	:	System error Malfunction of the kernel
E_CTX	:	Context error Called from a non-task context

[Definition of implementation dependency]
The behaviours of the system at an error according to the implementation definition of μITRON Ver 4.0 Specification are as follows in ASP:
① Called during CPU locked state.
An error message is output to syslog and the task is terminated after releasing CPU locked state.
② Called during dispatch prohibited state
An error message is output to syslog and the task is terminated after permitting dispatch.
③ Called during all non-priority masks released
An error message is output to syslog and the task if terminated after all priority mask are released.
④ Called from non-task part

The API is returned with E_CTX error issued.

[Difference from μITRON 4.0]
The specification is changed so that the type should be ER-type and that E_CTX error called from a non-task part should be returned

5.2.4. ter_tsk　Forcibly Terminate Other Tasks

[Function]
It forcibly terminates the specified task.
In concrete, if the task is not in dormant state, the process that should be done to the task at the task termination is performed and the target task becomes dormant.
Further, in case the number of the activation requests in queue of the target task is not 0, the process to be done to the task at the task activation is performed and the task becomes executable. At this time, the number of the activation requests in queue is decremented by 1.
The tasks to which this operation can be performed are in the case where the target and the own task (calling task) are allocated to the same processor. In case the allocated processor is different, E_OBJ is returned.

[Format]
ER ercd = ter_tsk(ID tskid);

[Parameter]
ID tskid　　:　ID number of the task subject to forced termination
* tskid cannot be specified with the ID number of the own task or a special macro TSK_SELF(=0) that specifies the own task.　If specified with such a value, E_ILSE error is returned and the API is returned.

[Return value]
ER ercd　:　Error code
E_OK　:　Completed normally
E_ID　:　Incorrect ID number (tskid is incorrect or cannot be used)
E_ILUSE　:　Service call incorrectly used (the own task is specified as the target task)
E_OBJ　:　Object state error (the target task is dormant)
The own task and the target task are allocated to different processors.
E_CTX　:　Context error
Called while CPU is locked
Called from non-task part.

[Definition of implementation dependency]
None.

5.2.5. get_pri Get Current Priority of Task

[Function]
It retrieves the current priority of the task

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

[Parameter]
ID tskid : Task ID number to be referenced
* In case TSK_SELF(=0) is specified, the own task becomes the target task.
PRI *p_tskpri : Pointer to the storage area of the current priority of the target task

[Return value]
ER ercd : Error code
E_OK : Terminated normally
E_ID : Incorrect ID number (tskid is incorrect or cannot be used)
E_NOEXS : Object not created (target task is not registered)
E_OBJ : Object state error (target task is dormant)
E_CTX : Context error
Called while CPU is locked
Called from non-task part.

[Definition of implementation dependency]
None.

5.2.6. chg_pri Change Base Priority of Task

[Function]
It prioritises the base priority of the task.
Accordingly, the current priority of the target task is also changed.

[Supplementary]
As ASP does not support mutex, the base priority and the current priority always match each other. For this reason, the internal management information does not include the base priority but it is managed with the current priority only.

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

[Parameter]
ID tskid : Task ID number to be referenced
* In case TSK_SELF(=0) is specified, the own task becomes the target task.
PRI tskpri : Change base priority of target task
* If TPRI_INI(=0) is specified, the base priority of the target task is changed to the initial priority at the time of activation.

[Return value]
ER ercd : Error code
- E_OK : Completed normally
- E_ID : Incorrect ID number (tskid is incorrect or cannot be used)
- E_PAR : Parameter (tskpri) is incorrect.
- E_CTX : Context error
  Called while CPU is locked
  Called from non-task part.
- E_OBJ : Object state error (target task is dormant)

[Definition of implementation dependency]
None.

[Relation with µITRON 4.0 Specification]
The condition where the target task is put in the lowest priority of all the tasks with the same priority has been changed. (if the target task is in waiting state and linked to a queue in the order of the task priority, the order is placed in the last of the tasks with the same priority)

5.2.7. get_inf Reference Extension Information of Own Task

[Function]
It references the extension information of the own task specified at creation of the task. The extension information referenced is returned to the memory area specified with p_extinf.

[Format]
ER ercd = get_inf(intptr_t *p_exinf);

[Parameter]
intptr_t *p_exinf : Pointer to memory area where extension information is stored

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
Called while CPU is locked
Called from non-task part.

[Relation with µITRON 4.0 Specification]
It is a newly introduced API that does not exist in µITRON 4.0 Specification.

5.2.8. ref_tsk Reference Task State

[Function]
It references the current state of the task (target task) specified with tskid. The current state referenced is returned to the memory area specified with pk_rtsk. If TSK_SELF (=0) is specified in tskid, the own task becomes the target task.
ref_tsk is a function for debugging and is not used for any other purpose. The reason is that, if ref_tsk is called and an interrupt occurs immediately after reference to the current state of the target task, the state of the target task may have been changed when ref_tsk is returned.

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

[Parameter]
ID tskid : ID number of target task
T_RTSK *pk_rtsk : Pointer to packet where the task current state is placed

T_RTSK to be returned has the following information:

STAT tskstat	Task state
PRI tskpri	Current priority of task
PRI tskbpri	Base priority of task
STAT tskwait	Task waiting factor
ID wobjid	Object ID of object of waiting task
TMO lefttmo	Time until task times out.
uint_t actcnt	Number of activation requests of task in queue
uint_t wupcnt	Number of wake-up requests of tasks in qeueu
ID prcid	ID of processor to which task is allocated
ID actprc	ID of processor to which task is allocated at next activation

One of the following values representing the current task state of the target task is returned:

TTS_RUN	0x01U	Execution state
TTS_RDY	0x02U	Executable state
TTS_WAI	0x04U	Waiting state
TTS_SUS	0x08U	Suspended state
TTS_WAS	0x0cU	Waiting and Suspended duplicate state
TTS_DMT	0x10U	Dormant state

In case the target task is in waiting state, one of the following values representing what the target task is waiting for is returned:

TTW_SLP	0x0001U	Waiting for wake-up
TTW_DLY	0x0002U	Waiting for time to pass
TTW_SEM	0x0004U	Waiting for semaphore to acquire the resource
TTW_FLG	0x0008U	Waiting for event flag
TTW_SDTQ	0x0010U	Waiting for sending to data queue
TTW_RDTQ	0x0020U	Waiting for reception from data queue
TTW_SPDQ	0x0100U	Waiting for sending to priority data queue
TTW_RPDQ	0x0200U	Waiting for reception from priority data queue
TTW_MBX	0x004.0U	Waiting for reception from mailbox
TTW_MPF	0x2000U	Waiting for seizure of fixed-length memory block

If the target task is not in waiting state, the value tskwait is not guaranteed.
If the target task is waiting for wake-up or in other waiting state than waiting for time to pass, the ID number of the object that the target task is waiting for is returned to wobjid.
In case the target task is not in waiting state, or it is waiting for wake-up or for time to pass, the value of wobjid is not guaranteed.

If the target task is waiting for any other event than time to pass, the relative time till the task times out is returned to lefttmo.
If the task does not time out, TMO_FEVR (=-1) is returned.
If the target task is waiting for time to pass, the relative time until the waiting is released after elapsing the delay time of the task is returned to lefttmo.
In some cases, however, the relative time to be returned cannot be stored in TMO type. In that case, the value that is type-cast to the relative time (defined to RELTM type, unit_t type) is returned.
In case the target task is not in waiting state, leftmo value is not guaranteed.

The number of the activation requests in queue of the target task is returned to actcnt.
If the target task is not in dormant state, the number of the wake-up request in queue of the target task is returned to wupcnt. If the target task is in dormant state, wupcnt value is not guaranteed.

Processor ID to which the target task is allocated is returned to prcid. Processor ID to be allocated at the next activation time is returned to actprc. If the processor to be allocated at the next activation time is not set, TPRC_NONE(=0) is returned.

[Return value]
ER ercd : Error code
- E_OK : Completed normally
- E_CTX : Context error

Called from non-task context.
Called from CPU locked state

E_ID : Incorrect ID number
Outside the validity of tskid.

[Definition of implementation dependency]
None.

5.3. Task-associated Synchronisation Function

5.3.1. slp_tsk/tslp_tsk　Wait for Task to Wake-up

[Function]
It puts the own task to waiting for wake-up state.
If the number of wake-up requests in queue is 0, the own task is placed in waiting for wake-up state.
If the number of wake-up requests in queue is not 0, the number of the wake-up requests is decremented by 1.
In this case, the own task maintains the executable state and the priority of the own task does not change.

[Format]
No timeout.　:　ER ercd = slp_tsk();
With timeout　:　ER ercd = tslp_tsk(TMO tmout);

[Parameter]
TMO tmout　:　Timeout time (in 1 ms)
It specifies the timeout time in 1 ms unit.
If the timeout time is specified with TMO_POL (=0), the task will not be put in sleeping state and returned immediately with the timeout error (E_TMOUT).
If TMO_FEVR (=1) is specified, it waits for wake-up without timeout.
It is the same operation as the case when slp_tsk() is called.

[Return value]
ER ercd　:　Error code
- E_OK　:　Completed normally
- E_PAR　:　Parameter error (tmout is incorrect)
- E_RLWAI　:　Waiting state forcibly released (rel-wai accepted during waiting state)
- E_TMOUT　:　Timeout has occurred
- E_CTX　:　Context error
  Called in the following dispatched suspended state:
  - ・Called while CPU is locked.
  - ・Called while all the non-interrupt priority masks are released.
  - ・Called from non-task part.
  - ・Called while dispatch is prohibited.

[Definition of implementation dependency]
None.

5.3.2. wup_tsk/iwup_tsk Wake Up Task

[Function]
It wakes up the specified task.
If the target task is in waiting state, waiting of the target task is released.
The service call put into waiting state returns E_OK to the task released from waiting.
If the target task is not in waiting for wake-up state and not in sleeping state, the number of the wake-up requests in queue of the target task is increments by 1. If 1 is added to the number of the wake-up requests in queue and it exceeds TMAX_WUPCNT, it results in E_QOVR error.
If tskid is specified with TSK_SELF (=0) in wup_tsk, the own becomes the target task.

[Format]
Call from task part : ER ercd = wup_tsk(ID tskid);
Call from non-task part : ER ercd = iwup_tsk(ID tskid);

[Parameter]
ID tskid : ID number of task to be woken up
If TSK_SELF (=0) is specified, the own task becomes the target task.

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (tskid is incorrect or cannot be used)
E_NOEXS : Object not created (target task is not registered)
E_CTX : Context error
Called while CPU is locked
wup_tsk was called from non-task part.
iwup_tsk was called from task part
E_OBJ : Object state error (the target task is dormant)
E_QOVR : Queuing overflow (Overflow of wake-up requests in queue)

[Definition of implementation dependency]
The number of the wake-up requests in queue is fixed to 1 and the wake-up request is suspended once.

5.3.3. can_wup　Cancel Task Wake-up Request

[Function]
It cancels a wake-up request of a task.
It cancels all the wake-up requests that are not processed to the task specified with tskid (target task) and returns the number of the wake-up tasks cancelled.
If the target is not in sleeping state, the number of the wake-up requests in queue of the target is set to 0 and the number of the wake-up requests in queue before being set to 0 is returned as the return value of the service call.
If TSK_SELF (=0) is specified in tskid, the own task becomes the target task.

[Format]
ER_UINT actcnt = can_wup(ID tskid);

[Parameter]
ID tskid　　:　Task ID number whose task activation request is to be cancelled.
If TSK_SELF (=0) is specified, the own task becomes the target task.

[Return value]
ER_UINT actcnt　:　The number of times of the activation requests in queue (0 or more), or error code.
　　Value equal to or more than zero.　Number of times of activation requests in queue
　　E_ID　　:　Incorrect ID number (tskid is incorrect or cannot be used)
　　E_OBJ　　:　Object state error (the target task is dormant)
　　E_CTX　　:　Context error
　　Called while CPU is locked
　　Called from non-task part.

[Definition of implementation dependency]
None.

5.3.4. rel_wai/irel_wai Forcibly Release from Task-Waiting State

[Function]
It releases forcibly the task in waiting state. The task released from waiting state is returned with E_RLWA error.
If the target task in waiting state, the target task is released from waiting state.
To the task released from waiting state, E-RLWAI is returned from the service call that is placed in waiting state.
If tskid is specified with TSK_SELF(=0), it results in E-ID.

[Format]
Call from task part : ER ercd = rel_wai(ID tskid);
Call from non-task part : ER ercd = irel_wai(ID tskid);

[Parameter]
ID tskid : ID number of task to be forcibly released from waiting state

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (tskid is incorrect or cannot be used)
E_OBJ : Object state error (the target task is not in waiting state)
E_CTX : Context error
Called while CPU is locked
rel_wai was called from non-task part.
irel_wai was called from task part.

5.3.5. sus_tsk Suspend Task

[Function]
It forcibly moves a task to suspended state
It forcibly makes a task specified with tskid (target task) wait. In concrete, when the target task is in executable state, the target task enters in forced Suspended state.
If the target task is in waiting state (excluding waiting in duplicate), it enters in waiting and suspended duplicate state.
A suspend request can not nest.
If the target task is already in suspended state or waiting and suspended dupulicate state, E_QOVR is returned.
If TSK_SELF (=0) is specified in tskid, the own task becomes the target task.

[Format]
ER ercd = sus_tsk(ID tskid);

[Parameter]
ID tskid : ID number of task to be moved
If TSK_SELF (=0) is specified in tskid, the own task becomes the target task.

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (tskid is incorrect or cannot be used)
E_OBJ : Object stauts error (target task is dormant)
E_QOVR : Queuing overflow (Overflow of the number of forced request nests)
E_CTX : Context error
Called while CPU was locked.
Called from non-task part.
If sus_tsk is called with the target task as the own task while dispatch is suspended, it results in E_CTX error.

[Definition of implementation dependency]
The number of the forced waiting requests to be nested is 0 and the number of the requests is not nested.

5.3.6. rsm_tsk　Resume Task From Suspended State

[Function]
It resumes the task from the forced suspended state.
If suspending is released while the task is in waiting and suspended in duplicate, it enters in the normal waiting state.
If tskid is specified with TSK_SELF(=0), it results in E_ID.

[Format]
ER ercd = rsm_tsk(ID tskid);

[Parameter]
ID tskid　　:　ID number of task to be resumed

[Return value]
ER ercd　　:　Error code
　　E_OK　　:　Completed normally
　　E_ID　　:　Incorrect ID number (tskid is incorrect or cannot be used)
　　E_NOEXS　　:　Object not created (target task is not registered)
　　E_OBJ　　:　Object state error (target task is not in forced waiting state)
　　E_CTX　　:　Context error
　　Called while CPU is locked
　　Called from non-task part.

[Definition of implementation dependency]
None.

5.3.7. frsm_tsk　Forcibly resume task from suspended state (Abolished)

[Difference from µITRON 4.0]
The maximum number of suspend request nests is fixed to 1. Consequently, the service call that forcibly resumes a task from suspending (frsm_tsk) has been abolished.

5.3.8. dly_tsk Delay Execution of Own Task

[Function]
It delays execution of the own task for the specified period of time.

[Format]
ER ercd = dly_tsk(RELTIM dlytim);

[Parameter]
RELTIM dlytim : Period of time for which own task is delayed (relative time (in 1 ms))

[Return value]
ER ercd : Error code
- E_OK : Completed normally
- E_PAR : Parameter error (dlytim is incorrect (larger tan TMAX_RELTIM))
- E_RLWAI : Waiting state forcibly released (rel-wai accepted during waiting state)
- E_CTX : Context error
  Called while dispatch is suspended.

[Definition of implementation dependency]
None.

5.4. Task Exception Handling Function

5.4.1. ras_tex/iras_tex Raise Task Exception Handling

[Function]
It requests task exception handling.
It requests a task exception request to a task specified with tskid (target task) for task exception cause specified with rasptn. The pending exception cause of the targe task is updated to bitwise OR ("|" in C-language) specified with rasptn.
A task exception handling routine is initiated when the following six conditions are fulfilled:

① Task exception handling is enabled;
② The pending exception cause is not 0;
③ The task is in execution state;
④ The task context is executed.
⑤ All the interrupt priority masks are released; and
⑥ CPU is not locked. When all these conditions are met, the task exception handling routine is initiated.

If tskid is specified with TSK_SELF(=0) in ras_tex, the own task becomes the target task.

[Format]
Called from task part : ER ercd = ras_tex(ID tskid, TEXPTN rasptn);
Called from non-task part : ER ercd = iras_tex(ID tskid, TEXPTN rasptn);

[Parameter]
ID tskid : ID number of task subject to request
TEXPTN rasptn : Task exception cause of task exception handling to be requested

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (tskid is incorrect or cannot be used)
E_NOEXS : Object not created (target task is not registered)
E_PAR : Parameter error (rapstn specified with 0)
E_OBJ : Object state error (the target task is in dormant state, or no task exception handling routine is defined for the target task.)
E_CTX : Context error
Called while CPU is locked
ras_tex was called from non-task part.
iras_tex was called from task part.

[Definition of implementation dependency]
None.

5.4.2. dis_tex Disable Task Exception Handling

[Function]
It disables task exception handling of the own task

[Format]
ER ercd = dis_tex(void);

[Parameter]
None.

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_OBJ : Object state error (No task exception handling routine is defined for the own task)
E_CTX : Context error
Called while CPU is locked
Called from non-task part.

[Definition of implementation dependency]
None.

5.4.3. ena_tex Enable Task Exception Handling

[Function]
It enables the own task for task exception handling.

[Supplementary]
A task exception handling routine is called in task exception handling disabled state, but, by calling ena_tex during a task exception handling routine, task exception handling routines can be activated in multiple.
In consequence, multiple activation of task exception handling routines may result in consumption of the stack area for the task and eventually in stack overflow.
The kernel does not restrict the maximum level of multiple activation. It is up to the application that restricts the maximum level of multiple activation.

[Format]
ER ercd = ena_tex(void);

[Parameter]
None.

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_OBJ : Object state error (No task exception handling routine is defined for the own task)
E_CTX : Context error
Called while CPU was locked.
Called from non-task part.

[Definition of implementation dependency]
None.

5.4.4. sns_tex　Reference Task Exception Handling Disabled State

[Function]
It returns the task exception handling disabled state of the task in execution state.
This service call can be called from either a task part or a non-task part.

[Format]
BOOL state = sns_tex(void);

[Parameter]
None.

[Return value]
BOOL state　:　Task exception handling disabled state
　TRUE　:　Task exception handling is disabled for the task in execution state
　FALSE　:　Task exception handling is enabled for the task in execution state

[Definition of implementation dependency]
None.

5.4.5. ref_tex　Reference Task Exception Handling State

[Function]
It references the current state regarding the task exception handling of the task specified with tskid (target task). The current state referenced is returned in the packet specified in pk_rtex. Either of the following values provided below that represents the current task exception handling disabled flag of the target task is returned to texstat.

TTEX_ENA　　0x01U　Task exception handling enabled
TTEX_DIS 0x02U　Task exception handling disabled

The current pending exception cause of the target task is returned to pndptn
If TSK_SELF (=0) is specified in tskid, the own task becomes the target task.

[Format]
ER ercd = ref_tex(ID tskid, T_RTEX *pk_rtex);

[Parameter]
ID tskid　　:　ID number of target task
T_RTEX *pk_rtex　:　Pointer to the packet in which the current state of task exception handling is placed.
* T_RTEX that is returned contains the following information:

STAT	texstat	Task exception handling state
TEXPTN	pndptn	Task pending exception cause

[Return value]
ER ercd　:　Error code

- E_OK　:　Completed normally
- E_CTX　:　Context error
  - ・Called from non-task context.
  - ・Called while CPU was locked
- E_ID　:　Incorrect ID number
  - * pdqid is out of range

[Definition of implementation dependency]
None.

5.5. Semaphore Functions

5.5.1. sig_sem/isig_sem Return Semaphore Resource

[Function]
It returns semaphore resource.

[Format]
Call from task part : ER ercd = sig_sem(ID semid);
Call from non-task part : ER ercd = isig_sem(ID semid);

[Parameter]
ID semid : ID number of semaphore for which the resource is to be returned.

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (semid is incorrect or cannot be used)
E_NOEXS : Object no created (target semaphore is not registered)
E_QOVR : Queuing overflow (Returning of the number exceeding the maximum resources)
E_CTX : Context error
Called while CPU is locked
sig_sem was called from non-task part
isig_sem was called from task part

[Definition of implementation dependency]
None.

5.5.2. wai_sem/pol_sem/twai_sem Aquire Semaphore Resource

[Function]
It acquires a semaphore resource.

[Format]
With no timeout : ER ercd = wai_sem(ID semid);
Polling specified : ER ercd = pol_sem(ID semid);
With timeout : ER ercd = twai_sem(ID semid, TMO tmout);

[Parameter]
ID semid : ID number of semaphore for which the resource is to be acquired
TMO tmout : Specified timeout（in 1ms）

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (semid is incorrect or cannot be used)
E_NOEXS : Object no created (target semaphore is not registered)
E_RLWAI : Forced release of waiting state (rel_wai is accepted during waiting state except pol_sem)
E_TMOUT : Timeout or polling failed
E_CTX : Context error
Called while CPU is locked
Called from non-task part.
Called while dispatch is suspended (except pol_sem)
E_DLT : Re-initialisation of waiting object (except pol_sem)

[Definition of implementation dependency]
None.

5.5.3. ini_sem Re-initialise Semaphore

[Function]
It re-initialise a semaphore specified with semid.
The number of the resources of the target semaphore is initialised to the initial number of the resources. Also, the task bonded to the queue of the target semaphore is released gradually in the order from the task placed in the beginning of the queue.
E_DLT error is returned from the service call placed in waiting to the task released from waiting.

[Caution when using]
In case a semaphore is re-initialised, it is the responsibility of the application to maintain the conformity with the application.

[Format]
ER ercd = ini_sem(ID semid);

[Parameter]
ID semid : ID number of semaphore to be re-initialised

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
・Called from non-task context.
・Called while CPU was locked
E_ID : Incorrect ID number
* semid is out of range

[Definition of implementation dependency]
None.

5.5.4. ref_sem　Reference Semaphore State

[Function]
It references the current state of a semaphore specified with semid (target semaphore) The current state referenced is returned to the packet specified with pk_rsem.
In case no task exists in the target semaphore queue, TSK_NONE (=0) is returned to wtskid.

[Caution when using]
ref_sem is a function for debugging, use for any other purpose is not recommended. The reason is that, if ref_sem is called and an interrupt occurs immediately after reference to the current state of the target semaphore, the state of the target semaphore may have been changed when ref_sem is returned.

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

[Parameter]
ID psemid　　:　ID number of target semaphore
T_RSEM *pk_rsem :　Pointer to the packet in which the current state of semaphore is placed
T_RSEM to be returned contains the following information:
　ID wtskid　　:　ID number of the task in the beginning of the semaphore queue
　uint_t semcnt　:　Number of resources of semaphore

[Return value]
ER ercd　:　Error code
　E_OK　　:　Completed normally
　E_CTX　　:　Context error
　　・Called from non-task context.
　　・Called while CPU was locked
　E_ID　　:　Incorrect ID number
　　* semid is out of range

[Definition of implementation dependency]
None.

5.6. Event Flag Function

5.6.1. set_flg/iset_flg Set Event Flag

[Function]
It sets an event flag.
It sets the bit specified with setptn of the event flag specified with flgid (target flag).
The bit pattern of the target event flag is updated to the bitwise OR ("|" in C-language) of the conventional value and the value specified with setptn.
In case the target flag is TA_WMUL attribute and tasks exist in the queue, tasks fulfilling the condition of release from waiting are released from waiting in the order from that bonded to the front of the queue.
The service call put into waiting state returns E_OK to the task released from waiting.
However, in case the target event flag is TA_CLR attribute, at the moment when one task fulfilling the condition of release from waiting is released, all the bit patterns of the target event flag are cleared to 0. So, other tasks will not be released from waiting.
If the target flag is TA_WMUL attribute and not TA_CLR attribute, more than one task may be released from waiting by set_flg or iset_flg.

[Format]
Call from task part : ER ercd = set_flg(ID flgid, FLGPTN setptn);
Call from non-task part : ER ercd = iset_flg(ID flgid, FLGPTN setptn);

[Parameter]
ID flgid : ID number of event flat to be set
FLGPTN setptn : Bit pattern to be set

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (flgid is incorrect or cannot be used)
E_CTX : Context error
Called while CPU is locked
set_flg was called from non-task part
iset_flg was called from task part.

[Definition of implementation dependency]
None.

5.6.2. clr_flg Clear Event Flag

[Function]
It clears an event flag.
It clears the bit specified with clrptn of the event flag specified with flgid (target event flag). The bit pattern of the target event flag is updated with the bitwise AND ("&" in C-language) of the current value and the bit-inverted value of the value specified with clrptn.

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

[Parameter]
ID flgid : ID number of event flat to be set
FLGPTN clrptn : Bit pattern to be cleared

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (flgid is incorrect or cannot be used)
E_NOEXS : Object not created (target event flag is not registered)
E_CTX : Context error
Called while CPU is locked
Called from non-task part.

[Definition of implementation dependency]
None.

5.6.3. wai_flg/pol_flg/twai_flg Wait for Event Flag

[Function]
It waits for the occurrence of an event flag. OR waiting condition and AND waiting condition can be specified as the waiting mode. In case the waiting release condition is fulfilled for a flag with TA_CLR attribute, all the events are cleared.

[Format]
With no timeout :
ER ercd = wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
Polling specified :
ER ercd = pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
Timeout specified :
ER ercd = twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);

[Parameter]
ID flgid : ID number of event flag to be waited for
FLGPTN waiptn : Waiting bit pattern
MODE wfmode : Waiting mode
TWF_ANDW : Wait until all waiting bit patterns are set.
TWF_ORW : Wait until one of waiting bit patterns is set
FLGPTN *p_flgptn : Pointer to the storage area of bit patterns when released
TMO tmout : Specified timeout（in 1ms）

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (flgid is incorrect or cannot be used)
E_NOEXS : Object not created (target event flag is not registered)
E_PAR : Parameter error (waiptn or wfmode is incorrect, tmout is invalid)
* waiptn is 0
* wfmode is invalid (not TWF_ORW or TWF_ANDW)
E_ILUSE : Incorrect use of service call (There is a task waiting with an event flag not TA_WMUL attribute)
E_RLWAI : Forced release from waiting (rel_wai is accepted during waiting, except pol_flg)
E_TMOUT : Timeout or polling failed
E_CTX : Context error
Called while CPU was locked.
Called from non-task part.
Called while dispatch was suspended (except pol_flg)
E_DLT : Delete or re-initialise waiting object (except pol_flg)

[Definition of implementation dependency]
None.

5.6.4. ini_flg Re-initialise Event Flag

[Function]
Re-initialise an event flag specified with flgid.
The bit pattern of the target event flag is initialised to the initial bit pattern.
Also, tasks bonded to the queue of the target event flag are released from waiting in the order from the task in the front of the queue.
E-DLT error is returned from the service call placed in waiting to the task released from waiting.

[Caution when using]
In case an event flag is re-initialised, it is the responsibility of the application to maintain the conformity with the application.

[Format]
ER ercd = ini_flg(ID flgid);

[Parameter]
ID flgid : ID number of event flag to be re-initialised

[Return value]
ER ercd : Error code
- E_OK : Completed normally
- E_CTX : Context error
  - ・Called from non-task context.
  - ・Called while CPU was locked
- E_ID : Incorrect ID number
  - ・flgid is out of range.

[Definition of implementation dependency]
None.

5.6.5. ref_flg　Reference Event Flag State

[Function]
It references the current state of the event flag specified with flgid (target event flag).
The current state referenced is returned to the packet specified with pk_rflg.
In case no task exists in the target event flag, TSK_NONE (=0) is returned to wtskid.

[Caution when using]
ref_flg is a function for debugging and use of it for any other purpose is not recommended.
The reason is that, if ref_flg is called and an interrupt occurs immediately after reference to the current state of the event flag, the state of the event flag may have been changed when ref_flg is returned.

[Format]
ER ercd = ref_flg(ID flgid, T_RFLG *pk_rflg);

[Parameter]
ID flgid　　:　ID number of target event flag
T_RPDQ *pk_rflg　:　Pointer to the packet in which the current state of the event flag is placed
T_RFLG to be returned contains the following information:
　ID wtskid　:　ID number of the task in the front of the event flag queue
　uint_t flgptn　　:　Bit pattern of event flag

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_CTX　:　Context error
　　・Called from non-task context.
　　・Called while CPU was locked
　E_ID　:　Incorrect ID number
　　・flgid is out of range.

[Definition of implementation dependency]
None.

5.7. Data Queue Function

5.7.1. snd_dtq/psnd_dtq/ipsnd_dtq/tsnd_dtq Send to Data Queue

[Function]
It sends data to data queue. If there is no vacancy in the data queue, it waits until a vacancy occurs.
In case of sending to a data queue with size 0 while there is no waiting task, the operations is placed in waiting state until the waiting task occurs.

[Format]
Waiting time not specified :
ER ercd = snd_dtq(ID dtqid, intptr_t data);
Polling specified, call from task part :
ER ercd = psnd_dtq(ID dtqid, intptr_t data);
Polling specified, call from non-task part :
ER ercd = ipsnd_dtq(ID dtqid, intptr_t data);
With timeout :
ER ercd = tsnd_dtq(ID dtqid, intptr_t data, TMO tmout);

[Parameter]
ID dtqid : ID number of data queue to which data is sent
intptr_t data : Data to be sent to data queue
TMO tmout : Time of timeout (in case of tsnd_dtq)（in 1ms）

[Return value]
ER ercd : Error code
E_OK : Terminated normally
E_CTX : Context error
・Called from non-task context (except ipsnd_dtq)
・Called from task context (except ipsnd_dtq)
・Called while CPU was locked
・Called while dispatch was suspended (in case of snd_dtq and tsnd_dtq)
E_ID : Incorrect ID number
・dtqit is out of range
E_PAR : Parameter error
* tmout is invalid (in case of tsnd_dtq)
E_TMOUT : Polling failed or timeout (except snd_dtq)
E_RLWAI : Forced released from waiting disabled state or waiting state (in case of snd_dtq and tsnd_dtq)
E_DLT : Deletion or re-initialisation of waiting object (in case of snd_dtq and tsnd_dtq)

[Definition of implementation dependency]
None.

[Relation with μITRON 4.0 Specification]
The type of data to be sent has been changed from VP_INT to intprt_t type.

5.7.2. fsnd_dtq/ifsnd_dtq Forcibly Send to Data Queue

[Function]
It forcibly sends to data queue.
With normal data sending, if there is no vacancy in the data queue and there is no task waiting for reception of data in that data queue, the calling task enters in waiting for sending state. However, forced sending does not wait.
The concrete process is shown below:

① If a waiting task exists for the data queue, the function passes the sending data to the front task in the reception waiting queue and releases the waiting state.
② If there is no task waiting for the data queue exists, it adds the sending data to the end of the data queue. If the data queue is full, it deletes the front data and adds the data in the end of the queue.

This API will not enter in waiting state internally.
Therefore, it cannot be used for a data queue with capacity size 0. If called, error of incorrect use (E_ILUSE) is returned.

[Format]
Call from task part : ER ercd = fsnd_dtq(ID dtqid, intptr_t data);
Call from non-task part : ER ercd = ifsnd_dtq(ID dtqid, intptr_t data);

[Parameter]
ID dtqid : ID number of data queue to which data is sent
intptr_t data : Data to be sent to data queue

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (dtqid is incorrect or cannot be used)
E_ILUSE : Incorrect use of service call (The data area of the target data queue has capacity 0)
E_CTX : Context error
Called while CPU is locked
fsnd_dtq was called from non-task part
ifsnd_dtq was called from task part.

[Definition of implementation dependency]
None.

[Relation with µITRON 4.0 Specification]
The type of data to be sent has been changed from VP_INT to intprt_t type.

5.7.3. rcv_dtq/prcv_dtq/trcv_dtq Receive from Data Queue

[Function]
It receives data from a data queue.
If there is no data in the data queue, it waits until data occurrs.
If receiving from a data queue with size 0, it waits until a sending task occurs.

[Format]
Waiting time not specified : ER ercd = rcv_dtq(ID dtqid, intptr_t *p_data);
Polling specified : ER ercd = prcv_dtq(ID dtqid, intptr_t *p_data);
With timeout : ER ercd = trcv_dtq(ID dtqid, intptr_t *p_data, TMO tmout);

[Parameter]
ID dtqid : ID number of data queue to be received from
intptr_t *p_data : Pointer to the storage area of the data received from the data queue
TMO tmout : Time of timeout (in case fo trcv_dtq)（in 1ms）

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
・Called from non-task context
・Called while CPU was locked
・Called while dipatch was suspended (except prcv_dtq)
E_ID : Incorrect ID number
・dtqit is out of range
E_PAR : Parameter error
・tmout is invalid (in case of trcv_dtq)
E_TMOUT : Polling failed or timeout (except rcv_dtq)
E_RLWAI : Forced release from waiting disabled stauts or waiting state (except prcv_dtq)
E_DLT : Deletion or re-initialisation of waiting object (except prvc_dtq)

[Definition of implementation dependency]
None.

[Relation with μITRON 4.0 Specification]
The type of data to be sent has been changed from VP_INT to intprt_t type.

5.7.4. ini_dtq Re-initialise Data Queue

[Function]
It re-initialises a data queue.
The data queue management area of the target data queue is initialised to the state where there is no stored data.
The task bonded to the send queue and the receive queue of the target data queue is released from waiting in the order from the task in the front of each of the queues.
E-DLT error is returned from the service call placed in waiting to the task released from waiting.

[Caution when using]
In case a data queue is re-initialised, it is the responsibility of the application to maintain the conformity with the application.

[Format]
ER ercd = ini_dtq(ID dtq id);

[Parameter]
ID dtq id : Id number of priority data queue to be initialised

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
・Called from non-task context.
・Called while CPU was locked
E_ID : Incorrect ID number
* dtqid is out of range.

[Definition of implementation dependency]
None.

5.7.5. ref_dtq Reference Data Queue State

[Function]
It references the current state of a data queue specified with dtqid (target data queue). The current state referenced is returned to the packet specified with pk_rdtq. If there is no task existing in the send queue of the target data queue, TSK_NONE (=0) is returned to stskid.
If there is no task existing in the receive queue, TSK_NONE (=0) is returned to rtskid.

[Caution when using]
ref_dtq is a function to be used for debugging, use of it for any other purpose is not recommended.
The reason is that, if ref_dtq is called and an interrupt occurs immediately after reference to the current state of the data queue, the state of the data queue may have been changed when ref_dtq is returned.

[Format]
ER ercd = ref_ dtq (ID dtq id, T_RDTQ *pk_rtdq);

[Parameter]
ID dtq id : ID number of target data queue
T_RDTQ *pk_r dtq : Pointer to the packet in which the current state of the data queue is placed.
T_RDTQ to be returned contains the following information:
ID stskid : ID number of the task in the front of the send queue of the data queue
ID rtskid : ID number of the task in the front of the receive queue of the data queue.
uint_t sdtqcnt : Number of data stored in the data queue storage area

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
・Called from non-task context.
・Called while CPU was locked
E_ID : Incorrect ID number
* dtqid is out of range.

[Definition of implementation dependency]
None.

5.8. Priority Data Queue Function

Priority data queue is a newly introduced object not defined in "μITRON 4.0 Specification"

5.8.1. snd_pdq/psnd_pdq/tsnd_pdq/ipsnd_pdq Send to Priority Data Queue

[Function]
It sends data to the priority data queue with the specified priority.
If tasks exist in the receive queue of the target priority data queue, the task in the front of the receive queue receives the data specified with data and then is released from waiting.
The service call put into waiting state returns E_OK to the task released from waiting.
If no task exists in the receive queue of the target priority data queue and there is space in the priority data queue management area where data can be stored, the data specified with data is stored in the priority data queue management area with the priority specified with datapri.
If no task exists in the receive queue of the target priority data queue and there is no space in the priority data queue management area where data can be stored, the own task enters in the sending waiting state for the priority data queue and is bonded to the send queue of the target data queue.
datapri shall be equal to or more than TMIN_DPRI and the maximum level of data priority that can be sent to the target data queue shall be equal to or less than the maximum value. Otherwise, it results in E_PAR.
If there is no vacancy in the priority data queue, it waits until a vacancy occurs.
In case of sending to a data queue with size 0 while there is no waiting task, the operations is placed in waiting state until the waiting task occurs.

[Format]
Task part:
Sending to priority data queue
ER ercd = snd_pdq(ID pdqid, intptr_t data, PRI datapri);
Sending to priority data queue (polling)
ER ercd = psnd_pdq(ID pdqid, intptr_t data, PRI datapri);
Sending to priority data queue (with timeout)
ER ercd = tsnd_pdq(ID pdqid, intptr_t data, PRI datapri, TMO tmout);
Non-task part:
Sending to priority data queue (polling)
ER ercd = ipsnd_pdq(ID pdqid, intptr_t data, PRI datapri);

[Parameter]

ID pdqid	:	ID number of priority data queue to be sent to
intptr_t data	:	Data to be sent to data queue
PRI datapri	:	Priority of sending data
TMO tmout	:	Specified timeout（in 1ms）

[Return value]

ER ercd : Error code

E_OK : Completed normally

E_CTX : Context error

・snd_pdq, psnd_pdq or tsnd_pdq was called from non-task context

・ipsnd_pdq was called from task context.

・Called while CPU was locked.

・snd_pdq and tsnd_pdq were called while dispatch was suspended.

E_ID : Incorrect ID number

* pdqid is out of range

E_PAR : Parameter error

・tmout is invalid (in case of tsnd_pdq)

E_TMOUT : Polling failed or timeout (in case of psnf_pdq, ipsnd_pdq or tsnd_pdq)

E_RLWAI : Forced release of waiting disabled state of waiting state (In case of snd_pdq and tsnd_pdq)

E_DLT : Deletion or re-initialisation of waiting object

[Definition of implementation dependency]

None.

5.8.2. rcv_pdq/prcv_pdq/trcv_pdq Receive from Priority Data Queue

[Function]
It receives data from the priority data queue (target data queue) specified with pdqid. The data received is returned to the memory area specified with p_data and its priority is returned to the memory area specified with p_datapri.
If data is stored in the priority data queue management area of the target priority data queue, data is retrieved that is stored in the front of the priority data queue management area and returned to the memory area specified with p_data. In addition, the priority is returned to the memory area specified with p_datapri.
Further, if tasks exist in the send queue, the sending data of the task in the front of the send queue is stored in the priority data queue management area in the order of the data priority and then the task is released from waiting. The service call put into waiting state returns E_OK to the task released from waiting.
If no data is stored in the priority data queue management area of the target priority data queue and tasks exist in the send queue, the sending data of the task in the front of the send queue is returned to the memory area specified with p_data.
In addition, the priority is returned to the memory area specified with p_datapri.
The task in the front of the send queue is released from waiting. To the task released from waiting, E-OK is returned from the service call that is put into waiting state.
If no data is stored in the priority data queue management area of the target priority data queue and no task exists in the send queue, the own task enters in waiting state for data sent from the priority data queue and is bonded to the receive queue of the target priority data queue.
If there is no data in the priority data queue, it waits until data occurs.
If receiving from a data queue with size 0, it waits until a sending task occurs.

[Format]
Receiving from priority data queue
```
ER ercd = rcv_pdq( ID pdqid, intptr_t *p_data, PRI *p_datapri );
```
Receiving from priority data queue (polling)
```
ER ercd = prcv_pdq( ID pdqid, intptr_t *p_data, PRI *p_datapri );
```
Receiving from priority data queue (with timeout)
```
ER ercd = trcv_pdq( ID pdqid  intptr_t *p_data, PRI *p_datapri, TMO tmout );
```

[Parameter]

ID pdtqid	:	ID number of priority data queue from which data is received
intptr_t *p_data	:	Pointer to the storage area of the data received from the data queue
PRI *p_datapri	:	Pointer to the priority storage area of the data received from data queue.
TMO tmout	:	Specified timeout（in 1ms）

[Return value]

ER ercd : Error code

E_OK : Completed normally

E_CTX : Context error

・Called from non-task context.

・Called while CPU was locked

・Called while dispatch was suspended (except prcv_pdq)

E_ID : Incorrect ID number

・ pdqid is out of range [NGKI1884]

E_PAR : Parameter error

・tmout is invalid (in case of trcv_pdq) [NGKI1885]

E_TMOUT : Polling failed or timeout (except rcv_pdq)

E_RLWAI : Forced release from waiting disabled state or waiting state (except prcv_pdq)

E_DLT : Deletion or re-initialisation of waiting object (except prcv_pdq)

[Definition of implementation dependency]

None.

5.8.3. ini_pdq Re-initialise Priority Data Queue

[Function]
It re-initialises a priority data queue.
The priority data queue management area of the target priority data queue is initialised to the state containing no data.
Tasks bonded to the send queue and the waiting queue of the target priority data queue are released from waiting from the task in the front of the respective queue. E-DLT error is returned from the service call placed in waiting to the task released from waiting.

[Caution when using]
If the priority data queue is re-initialised, it is the responsibility of the application to maintain the conformity with the application.

[Format]
ER ercd = ini_pdq(ID pdqid);

[Parameter]
ID number of priority data queue to be re-initialised

[Return value]
ER ercd : Error code
- E_OK : Completed normally
- E_CTX : Context error
  - ・Called from non-task context.
  - ・Called while CPU was locked
- E_ID : Incorrect ID number
  - * pdqid is out of range

[Definition of implementation dependency]
None.

5.8.4. Reference Priority Data Queue State

[Function]
It references the current state of the priority data queue specified with pdqid (target priority data queue).
The current state referenced is returned to the packet specified with pk_rpdq.
If no task exists in the send queue, TSK_NONE (=0) is returned to stskid. If there is no task existing in the receive queue, TSK_NONE (=0) is returned to rtskid.

[Caution when using]
ref_pdq is a function to be used for debugging and use of it for any other purpose is not recommended The reason is that, if ref_pdq is called and an interrupt occurs immediately after reference to the current state of the target priority data queue, the state of the priority data queue may have been changed when ref_pdq is returned.

[Format]
ER ercd = ref_pdq(ID pdqid, T_RPDQ *pk_rpdq);

[Parameter]
ID pdqid : ID number of target priority data queue
T_RPDQ *pk_rpdq : Pointer to the packet in which the current state of the priority data queue is placed.
T_RPDQ to be returned contains the following information:
ID stskid : ID number of the task in the front of the send queue of the priority data queue
ID rtskid : ID number of the task in the front of the receive queue of the priority data queue
uint_t spdqcnt : The number of data stored in the priority data queue management area.

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
・Called from non-task context.
・Called while CPU was locked
E_ID : Incorrect ID number
* pdqid is out of range

[Definition of implementation dependency]
None.

5.9. Mailbox Function

5.9.1. snd_mbx Send to Mailbox

[Function]
Send to Mailbox
It sends the message specified with pk_msg to the mailbox specified with mbxid (target mailbox).
If tasks exist in the queue of the target mailbox, the task in the front of the queue receives the message specified with pk_msg and is released from waiting.
The service call put into waiting state returns E_OK to the task released from waiting.
If no task exists in the queue of the target mailbox, the message specified with pk_msg is bonded to the message queue in the order specified according to whether the mailbox attribute TA_MPRI is specified or not.
If the target mailbox is specified with TA_MRPI attribute, the value in msgpri field in the starting message header of the message specified with pk_msg must be larger than TMIN_MPRI and larger than the maximum value of the message priority that can be sent to the target mail box. Otherwise, it results in E_PAR error.

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

[Parameter]
ID mbxid : ID number of mailbox to be sent to
T_MSG *pk_msg : The starting address of the message packet to be sent to the mailbox
If the message with message priority is sent to the mailbox with TA_MPRI attribute (mailbox in the order of the priority), T_MSG_PRI structure shall be used instead of T_MSG and the message priority shall be configured to PRI msgpri.

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (mbxid is incorrect or cannot be used)
E_NOEXS : Object not created yet (target mailbox not registered yet)
E_PAR : Parameter error
A message the message priority of which in the message packet is incorrect (msgpri) was sent to the mailbox with TA_MPRI attribute.
E_CTX : Context error
Called while CPU was locked.
Called from non-task part.

[Definition of implementation dependency]
None.

5.9.2. rcv_mbx/prcv_mbx/trcv_mbx Receive from Mailbox

[Function]
It receives from mailbox. If there is no message in the mailbox, it waits until a message occurs.

[Format]

With no timeout :
ER ercd = rcv_mbx(ID mbxid, T_MSG **ppk_msg);
Polling specified :
ER ercd = prcv_mbx(ID mbxid, T_MSG **ppk_msg);
Timeout specified :
ER ercd = trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);

[Parameter]
ID mbxid : ID number of mailbox subject to receiving
T_MSG **ppk_msg : Pointer to the storage area of the starting address of the message packet received from the mailbox
If a message with the message priority is sent to the mailbox with TA_MPRI attribute (mailbox in the order of the priority), T_MSG_PRI** shall be cast and referenced instead of T_MSG.
TMO tmout : Time of timeout (in case fo trcv_mbx)（in 1ms）

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
・Called from non-task context
・Called while CPU was locked
・Cakked while dispatch is suspended (except prvc_mbx)
E_ID : Incorrect ID number
・mbtxid is out of range
E_PAR : Parameter error
・tmout is invalid (in case of trvc_mbx)
E_TMOUT : Polling failed or timeout (except rvc_mbx)
E_RLWAI : Forced release from waiting disabled state or waiting state (except prvc_mbx)
E_DLT : Re-initialisation of waiting object (except prvc_mbx)

[Definition of implementation dependency]
None.

5.9.3. ini_mbx Re-initialise Mailbox

[Function]
It re-initialises the mailbox specified with mbxid (target mailbox).
The mailbox management area of the target mailbox is initialised to the state where there is no message bonded to the message queue.
The tasks that were bonded to the queue of the target mailbox are released from waiting in the order from the task in the front of the queue. E-DLT error is returned from the service call placed in waiting to the task released from waiting.

[Caution when using]
If a mailbox is initialised, it is the responsibility of the application to maintain the comformity with the application.

[Format]
ER ercd = ini_mbx(ID mbx id);

[Parameter]
ID mbx id : ID number of the mailbox to be re-initialised

[Return value]
ER ercd : Error code
- E_OK : Completed normally
- E_CTX : Context error
  - ・Called from non-task context.
  - ・Called while CPU was locked
- E_ID : Incorrect ID number
  - ・mbx_id is out of vaild range

[Definition of implementation dependency]
None.

5.9.4. ref_mbx　Reference Mailbox State

[Function]
It references the current state of the mailbox specified with mbxid (target mailbox).
The current state referenced is returned to the packet specified with pk_rmbx.
If there is no task in the queue of the target mailbox, TSK_NONE (=0) is returned to wtskid.
Also, if there is no message bonded to the message queue, NULL is returned to pk_msg.

[Caution when using]
ref_mbx is a function to be used for debugging and use of it for any other purpose is not recommended, The reason is that, if ref_mbx is called and an interrupt occurs immediately after reference to the current state of the target mailbox, the state of the mailbox may have been changed when ref_mbx is returned.

[Format]
ER ercd = ref_mbx(ID mbx d, T_RMBX *pk_r mbx);

[Parameter]
ID mbx id　　:　ID number of target mailbox
T_RMBX *pk_rmbx　:　Pointer to the packet in which the current state of the mailbox is placed.
T_RMBX to be returned contains the following information:
　ID wtskid　　:　ID number of the task in the front of the queue of the mailbox
　T_MSG *pk_msg　:　The starting address of the message bonded to the front of the message queue.

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_CTX　:　Context error
　　・Called from non-task context.
　　・Called while CPU was locked
E_ID　　:　Incorrect ID number
　　・mbxid is out of range

[Definition of implementation dependency]
None.

5.9.5. Description of Message (T_MSG) Type

The header structure of a message is described hereunder.

(1) Header structure of message

```
typedef struct t_msg       /* message header of mailbox */
{
    struct t_msg *next;
} T_MSG;
```

(2) Header structure of message with message priority

```
typedef struct t_msg_pri   /* message header with priority */
{
    T_MSG msgque;    /* message header */
    PRI msgpri;             /* message priority 1～16 */
} T_MSG_PRI;
```

- The message format is compatible with µITRON Ver 4.0 Specification. This kernel uses a message header and does not queue a message. For the compatibility with µITRON Ver 4.0. the queuing area remains.
- If a message is sent to a mailbox with message priority (mailbox with TA_MPRI attribute), T_MSG_PRI structure shall be used as in µITRON Ver 4.0 and the message priority shall be specified. The range of the specifiable value as the message priority for this kernel is from 1 to 16.

5.10. Fixed-Length Memory Pool Function

5.10.1. get_mpf/pget_mpf/tget_mpf　Acquire Fixed-Length Memory Block

[Function]
It acquires a fixed-length memory block from the fixed-length memory pool specified with mpfid (target fixed-length memory pool) and returns the starting address to blk.
If there is a memory area not allocated yet where a fixed-length memory block can be allocated in the fixed-length memory pool of the target fixed-length memory pool, one fixed-length memory block is allocated and its starting address is returned to blk.
If there is no memory area not allocated yet, the own task enters in waiting state for acquiring a fixed-length memory block and own task is bonded to the waiting queue of the target fixed length memory pool.

[Format]
With no waiting time specified
　　ER ercd = get_mpf(ID mpfid, void **p_blk);
Polling specified
　　ER ercd = pget_mpf(ID mpfid, void **p_blk);
With timeout
　　ER ercd = tget_mpf(ID mpfid, void **p_blk, TMO tmout);

[Parameter]
ID mpfid　　:　ID number of the fixed-length memory pool to be acquired for memory block
void *p_blk　:　Pointer to the storage area of the starting address of the memory block acquired
TMO tmout　:　Specified timeout（in 1ms）

[Return value]
ER ercd　　:　Error code
　　E_OK　　:　Completed normally
　　E_ID　　:　Incorrect ID number (mpfid is incorrect or cannot be used)
　　E_CTX　　:　Context error
　　Called while CPU is locked
　　Called from non-task part.
　　Called while dispatch was suspended. (except pget_mpf)
　　E_TMOUT　　:　Polling failed or timeout (except get_mpf)
　　E_RLWAI　:　Forced release from waiting disabled state or waiting state (except pget_mfp)
　　E_DLT　　:　Deletion or re-initialisation of waiting object (except pget_mfp)

[Definition of implementation dependency]

None.

[Relation with μITRON 4.0 Specification]
The pointer variable to a memory block has been changed from VP to void pointer type.

5.10.2. rel_mpf　Release Fixed-Length Memory Block

[Function]
It releases the fixed-length memory block specified with blk to the fixed-length memory pool specified with mpfid (target fixed-length memory pool). The behaviour in concrete is as follows:
If tasks exist in the queue of the memory pool of the target fixed-length memory pool, the task in the front of the queue acquires the fixed-length memory block specified with blk and is released from waiting. The service call put into waiting state returns E_OK to the task released from waiting.
If there is no task in the queue, the fixed-length memory block specified with blk is returned to the memory area of the target fixed-length memory pool.
If blk is not the starting address of the fixed-length memory block acquired from the target fixed-length memory pool, it results in E_PAR error.

[Format]
ER ercd = rel_mpf(ID mpfid, void *blk);

[Parameter]
ID mpfid　:　ID number of fixed-memory pool to which the memory block is to returned
void *blk　:　The starting address of the memory block to be returned

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_ID　:　Incorrect ID number (mpfid is incorrect or cannot be used)
　E_NOEXS　:　Object not created yet (target fixed-length memory pool not registered yet)
　E_PAR　:　Parameter error (blk is incorrect, returned to a different memory pool or returned to any other address than the starting address of the memory block acquired.
　E_CTX　:　Context error
　　Called while CPU is locked
　　Called from non-task part.

[Definition of implementation dependency]
None.

[Relation with μITRON 4.0 Specification]
The pointer variable to a memory block has been changed from VP to void pointer type.

5.10.3. ini_MPF　Re-initialise Fixed-Length Memory Pool

[Function]
It re-initialises the fixed-length memory pool specified with mpfid (target fixed-length memory pool). The total memory pool area of the target fixed-length memory pool is initialised to the state not allocated yet.
Also, tasks bonded to the queue of the target fixe-length memory pool are released from waiting in the order from the task in the front of the queue. E-DLT error is returned from the service call placed in waiting to the task released from waiting.

[Caution when using]
If a fixed-length memory pool is re-initialised, it is the responsibility of the application to maintain the conformity with the application.

[Format]
ER ercd = ini_mpf(ID mpf id);

[Parameter]
ID mpf id　　:　ID number of fixed-length memory pool to be re-initialised

[Return value]
ER ercd　　:　Error code
　　E_OK　　:　Completed normally
　　E_CTX　　:　Context error
　　・Called from non-task context.
　　・Called while CPU was locked
　　E_ID　　:　Incorrect ID number
　　・mpf is out of range

[Definition of implementation dependency]
None.

5.10.4. ref_mpf　Reference Fixed-Length Memory Pool State

[Function]
It references the current state of the fixed-length memory pool specified with mpfid (target fixed-length memory pool). The current state referenced is returned to the packet specified with pk_rmpf. If there is no task in queue of the target fixed-length memory pool, TSK_NONE (=0) is returned to wtskid.

[Caution when using]
ref_mpf is a function to　be used for debugging and use of it for any other purpose is not recommended. The reason is that, if ref_mpf is called and an interrupt occurs immediately after reference to the current state of the target fixed-length memory pool, the state of the fixed-length memory pool may have been changed when ref_mpf is returned.

[Format]
ER ercd = ref_ mpf(ID mpf id, T_RMPF *pk_rmpf);

[Parameter]
ID mpf id　　:　ID number of target fixed-length memory pool
T_RMPF *pk_rmpf　　:　Pointer to the packet in which the current state of the fixed memory pool is placed
T_RMPF to　be returned contains the following information:
　ID wtskid　　:　ID number of the task in the front of the queue of the fixed-length memory pool
　uint_t fblkcnt　　:　The number of fixed-length memory blocks that can be allocated to the vacant memory area of the fixed-length memory pool.

[Return value]
ER ercd　　:　Error code
　E_OK　　:　Completed normally
　E_CTX　　:　Context error
　　・Called from non-task context.
　　・Called while CPU was locked
　E_ID　　:　Incorrect ID number
　　・mpf is out of range

[Definition of implementation dependency]
None.

5.11. Cyclic Handler Function

μITRON 4.0 Specification provides that it is not necessary to support the function of storing cycle phase (cyclic handler with TA_PHS attribute) in the standard profile. It is the same in New Generation Kernel Integration Specification. This kernel does not support the function of storing cycle phase.

5.11.1. sta_cyc　Start Cyclic Handler

[Function]
It starts the cyclic handler specified with cycid (target cyclic handler).
If the target cyclic handler is inactive, the target cyclic handler becomes active. The time to start the cyclic handler is set to the first activation time after sta_cyc is called.
The first activation time is the cycle phase time (cycphs) specified with the creation parameter from the time when this API is called.
If the target cyclic handler is active, the time when the cyclic handler starts next time can be re-configured.

[Supplementary]
For this kernel, which does not support TA_PHS attribute function, the time when the cyclic handler becomes active next time is set to the relative time specified with the cycle phase of the target cyclic handler (cycphs) from the time when sta_cyc is called.
You have to be careful because this part is different from μITRON Specification, where the cyclic handler shall be activated after the cycle time (cyctim).

[Format]
ER ercd = sta_cyc(ID cycid);

[Parameter]
ID cycid　:　ID number of cyclic handler to be started

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_ID　:　Incorrect ID number (cycid is out of range)
　E_CTX　:　Context error
　　Called while CPU was locked.
　　Called from non-task part.

[Definition of implementation dependency]
None.

[Relation with μITRON 4.0 Specification]

For a cyclic handler that does not have TA_PHS attribute, the time when the cyclic handler is activated for the first time since sta_cyc is called has been changed. μITRON 4.0 Specification defines that a cyclic handler is activated after the relative time specified with the cycle time (cyctim) of the cyclic handler has passed since sta_cyc was called. In New Generation Kernel Specification, on the other hand, a cyclic handler shall be active after the relative time specified with the cycle phase (cycphs).

5.11.2. stp_cyc　Stop Cyclic Hander

[Function]
It stops a cyclic handler.
It stops the cyclic handler specified with cycid (target cyclic handler).
If the target cyclic handler is active, it becomes inactive.
If the target cyclic handler is inactive, nothing is done and the operation is completed normally.

[Format]
ER ercd = stp _cyc(ID cycid);

[Parameter]
ID cycid　　:　ID number of cyclic handler to be stopped

[Return value]
ER ercd　　:　Error code
　　E_OK　　:　Terminated normally
　　E_ID　　:　Incorrect ID number (cycid is out of range)
　　E_CTX　　:　Context error
　　　　Called while CPU is locked
　　　　Called from non-task part.

[Definition of implementation dependency]
None.

5.11.3. ref_cyc　Reference Cyclic Handler State

[Function]
It references the current state of the cyclic handler specified with cycid (target cyclic handler). The current state referenced is returned to the packet specified with pk_rcyc.
To cycstat, one of the following values that represent the current operation state of a cyclic handler is returned:

TCYC_STP	0x01U	Cyclic handler is inactive
TCYC_STA	0x02U	Cyclic handler is active

If the target cyclic handler is active, the relative time until the next activation of the cyclic handler is returned to leftim.
If the target cyclic handler is inactive, the value of leftim is not guaranteed.

[Caution when using]
ref_cyc is a function to be used for debugging, use of it for any other purpose is not recommended. The reason is that, if ref_cyc is called and an interrupt occurs immediately after reference to the current state of the target cyclic handler, the state of the cyclic handler may have been changed when ref_cyc is returned.

[Format]
ER ercd = ref_cyc(ID cyc id, T_RCYC *pk_rcyc);

[Parameter]
ID pdqid　　:　ID number of target cyclic handler
T_RCYC *pk_rcyc　:　Pointer to the packet in which the current state of the cyclic handler is placed.
T_RCYC to be returned contains the following information:

STAT cycstat	:	Active state of cyclic handler
RELTIM lefttim	:	Relative time till cyclic handler is activated next time
ID prcid	:	ID of processor to which cyclic handler is allocated (invalid data for ASP)

[Return value]
ER ercd　:　Error code
E_OK　　:　Completed normally
E_CTX　　:　Context error
・Called from non-task context.
・Called while CPU was locked
E_ID　　:　Incorrect ID number
* pdqid is out of range

[Definition of implementation dependency]
None.

5.12. Alarm Handler Function

μITRON 4.0 Specification defines that it is not necessary to support this function.
In "TOPPERS New Generation Kernel Integration Specification", the alarm handler is included in the standard function set.

5.12.1. sta_alm/ista_alm　Start Alarm Handler

[Function]
It starts the operation of an alarm handler. If the target alarm handler is inactive, the target alarm handler becomes active.
The time to activate the alarm handler is set to the time after the relative time specified with almtim since sta_alm is called.
If the target alarm handler is active, the time to activate the alarm handler is re-configured only.

[Format]
Call from tasks context　　:　ER ercd = sta _alm(ID almid, RELTIM almtim);
Call from non-tasj context　　:　ER ercd = ista _alm(ID almid, RELTIM almtim);

[Parameter]
ID almid　　:　ID number of alarm handler
RELTIM almtim　:　Activation time of alarm handler

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_ID　:　Incorrect ID number (almid is out of range)
　E_PAR　:　Parameter error
　　・almtim is larger than TMAX_RELTIM.
　E_CTX　:　Context error
　　・Called from non-task context (in case of sta_alm)
　　・Called from task context (in case of ista_alm)
　　・Called while CPU is locked.

[Definition of implementation dependency]
None.

5.12.2. stp_alm/istp_alm Stop Alarm Handler

[Function]
It stops an active alarm handler.
If the target alarm handler is active, it becomes inactive.
If the target alarm handler is inactive, nothing occurs and the operation is completed normally.

[Format]
Call from task context : ER ercd = stp _alm(ID almid);
Call from non-task context : ER ercd = istp _alm(ID almid);

[Parameter]
ID almid : ID number of alarm handler

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_ID : Incorrect ID number (id is out of range)
E_CTX : Context error
・Called from non-task context (in case of stp_alm)
・Called from task context (in case of istp_alm)
・Called while CPU was locked.

[Definition of implementation dependency]
None.

5.12.3. ref_alm　Reference Alarm Handler State

[Function]
It references the current state of an alarm handler specified with almid. The current state referenced is returned to the packet specified with pk_alm.
Either of the following values representing the current operation state of the target alarm handler is returned to almstat.

TALM_STP	0x01U	Alarm handler inactive
TALM_STA	0x02U	Alarm handler active

If the target alarm handler is active, the relative time till the alarm handler is activated next time is returned to lefttim.
If the alarm handler is inactive, the value of lefttim is not guaranteed.

[Caution when using]
ref_alm is a function to be used for debugging and use of it for any other purpose is not recommended. The reason is that, if ref_alm is called and an interrupt occurs immediately after reference to the current state of the target alarm handler, the state of the alarm handler may have been changed when ref_alm is returned.

[Format]
ER ercd = ref_alm(ID almid, T_RALM *pk_ralm);

[Parameter]
ID almid　　:　ID number of the target alarm handler
T_RALM *pk_ralm　　:　Pointer to the packet in which the current state of alarm handler is palced
T_RALM to be returned contains the following information:

STAT almstat	:	Operation state of alarm handler
RELTIM lefttim	:	The relative time to the time when alarm handler is activated
ID prcid	:	ID of processor to which alarm handler is allocated (invalid for ASP)

[Return value]
ER ercd　　:　Error code

E_OK	:	Completed normally
E_CTX	:	Context error ・Called from non-task context. ・Called while CPU was locked
E_ID	:	Incorrect ID number ・almid is out of range

[Definition of implementation dependency]
None.

5.13. Time Management Function

5.13.1. get_tim　Reference System Time

[Function]
It references the system time.
The system time referenced is returned to the memory area specified with p_systim.

[Format]
ER ercd = get_tim(SYSTIM *p_systim);

[Parameter]
SYSTIM *p_systim　　:　Pointer to the storage area of the current system time

[Return value]
ER ercd　　:　Error code
　E_OK　　:　Terminated normally
　E_CTX　　:　Context error
　　Called while CPU is locked
　　Called from non-task part.

[Definition of implementation dependency]
None.

5.13.2. set_tim　Set System Time (Abolished)

[Difference from μITRON 4.0 Specification]
This API defined in μITRON 4.0 Specification has been removed in "TOPPERS New Kernel Integration Specification".

5.14. System State Management Function

5.14.1. get_tid/iget_tid　Get Task ID under Execution

[Function]
It references ID number of the task currently under execution (the own task in case of get_tid).
The task ID reference is returned to the memory area specified with p_tskid.
If there is no task under execution in case of iget_tid, TASK_NONE (=0) is returned.

[Format]
Call from task part　　:　ER ercd = get_tid(ID *p_tskid);
Call from non-task part　:　ER ercd = iget_tid(ID *p_tskid);

[Parameter]
ID *p_tskid　:　Pointer to the storage area of the task ID under execution
* If there is no task under execution in case of iget_tid, TSK_NONE is returned.

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_CTX　:　Context error
　　Called while CPU is locked
　　get_tid was called from non-task part.
　　iget_tid was called from task part.

[Definition of implementation dependency]
None.

5.14.2. loc_cpu/iloc_cpu　Transition to CPU Locked State

[Function]
It sets CPU locked flag and transitions to CPU locked state.
If this function is called while CPU is locked, nothing occurs, resulting in normal completion.

[Format]
Call from task part　　:　ER ercd = loc _cpu(void);
Call from non-task part　:　ER ercd = iloc_cpu(ID *p_tskid);

[Parameter]
None.

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_CTX　:　Context error
　　loc_pcu was called from non-task part.
　　iloc_pcu was called from task part.

[Definition of implementation dependency]
The method of realisation of CPU locked state varies depending on CPU.
See the manual of the corresponding CPU dependency part.

5.14.3. unl_cpu/iunl_cpu　Unlock CPU Locked State

[Function]
It unlocks CPU locked state of the system state.
It clears CPU locked flag and transitions to CPU unlocked state.
If this function is called while CPU is unlocked, nothing occurs, resulting in normal completion.

[Format]
Call from task part　　:　ER ercd = unl _cpu(void);
Call from non-task part　:　ER ercd = iunl_cpu(ID *p_tskid);

[Parameter]
None.

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_CTX　:　Context error
　　unl_cpu was called from non-task part.
　　iunl_cpu was called from task part.

[Definition of implementation dependency]
The method of realisation of CPU locked state varies depending on CPU.
See the manual of the corresponding CPU dependency part.

5.14.4. ena_dsp　Enable Dispatch

[Function]
It changes the system state so that dispatch of tasks is enabled.
IF this function is called while dispatch is enabled, nothing occurs, resulting in normal completion.
As a result of transition to dispatch-enabled state, dispatch-suspended state is released and a dispatch may take place.

[Format]
ER ercd = ena_dsp(void);

[Parameter]
None.

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_CTX　:　Context error
　　Called while CPU is locked
　　Called from non-task part.

[Definition of implementation dependency]
None.

5.14.5. dis_dsp　Disable Dispatch

[Function]
It changes the system state so that dispatch of tasks is disabled.
If this function is called while dispatch is disabled, nothing occurs, resulting in normal completion.

[Format]
ER ercd = dis_dsp(void);

[Parameter]
None.

[Return value]
ER ercd　　:　Error code
　　E_OK　　:　Completed normally
　　E_CTX　　:　Context error
　　　　Called while CPU is locked
　　　　Called from non-task part.

[Definition of implementation dependency]
None.

5.14.6. rot_rdq/irot_rdq　Rotate Task Execution Order

[Function]
It operates the task execution queue and rotates the task execution order.
The highest prioritised task of all the tasks executable having the priority specified with tskpri (target priority) is put to the lowest priority of all all the tasks with the same priority specified. If there is no task executable with the target priority or there is only one task, nothing occurs, resulting in normal completion.
If tskpri is specified with TPRI_SELF (=0) in rot_rdq, the base priority of the own task becomes the target priority.

[Format]
Call from task part　　:　ER ercd = rot_rdq(PRI tskpri);
Call from non-task part　:　ER ercd = irot_rdq(PRI tskpri);

[Parameter]
PRI tpri　　:　Priority of task to be rotated

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_PAR　:　Parameter error (tskpri is incorrect)
　E_CTX　:　Context error
　　Called while CPU was locked.
　　rot_rdq was called from non-task part.
　　irot_rdq was called from task part.

[Definition of implementation dependency]
None.

5.14.7. sns_xxx　Reference Different States

[Function]
It references the different states of the system.
The system states that can be referenced are as follows:

- ･Execution context state
- ･CPU locked state
- ･Dispatch disabled state
- ･Dispatch suspended state

This service call can be called from either task/non-task part,

[Format]
bool_t state = sns_xxx(void);

sns_ctx()　　:　Execution context state

If the execution context is in non-task part, TRUE is returned. If in task part, FALSE is returned.

sns_loc()　　: CPU locked state

If CPU is in locked state, TRUE is returned. If not locked, FALSE is returned.

sns_dsp()　　: Dispatch disabled state

TRUE is returned if dispatch is disabled. FALSE is returned if dispatch is enabled.

sns_dpn()　　: Dispatch suspended state

TRUE is returned if dispatch is suspended. Otherwise, FALSE is returned.

[Parameter]
None.

[Return value]
bool_t state　:　Determination flag of different state

[Definition of implementation dependency]
None.

[Relation with μITRON 4.0 Specification]
The return value has been changed from BOOL type to boot_t type.

5.15. Interrupt Management Function

5.15.1. ena_int () Enable Interrupt

[Function]
It enables an interrupt by the interrupt source specified with the interrupt number.
It clears the interrupt request disabled flag of the interrupt request line specified with intno (target interrupt request line).
If the interrupt request disabled flag of the target interrupt request line cannot be cleared because of the definition of the target, it results in E_PAR error.
In concrete, setting an interrupt request disabled flag to the target interrupt request line may not be supported.
ena_int may not be supported due to the definition of the target.
If ena_int is supported, TOPPERS_SUPPORT_ENA_INT is defined as macro. If not supported, when ena_int is called, E_NOSPT error is returned or an error occurs when linking.

[Format]
ER errcd = ena_int(INTNO intno);

[Parameter]
INTNO intno : Interrupt number to be enabled

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
・Called from non-task context
E_NOSPT : Not supported error
E_PAR : Parameter error
・intno is out of range.
・The interrupt request disabled flag of the target interrupt request line cannot be cleared because of the definition of the target.
E_OBJ : Object error
・The interrupt request line attribute is not configured to the target interrupt request line.

[Relation with μITRON 4.0 Specification]
intno, which was implementation-defined in μITRON 4.0 Specification, has been standardised.
In TOPPERS New Generation Kernel Integration Specification, it can be called while CPU is locked.

5.15.2. dis_int()　Disable Interrupt

[Function]
It disables an interrupt by the interrupt source specified with the interrupt number.
It sets the interrupt request disabled flag to the interrupt request line specified with intno (target interrupt request line). If the interrupt request disabled flag of the target interrupt request line cannot be set, it results in E_PAR error.
In concrete, setting an interrupt request disabled flag to the target interrupt request line may not be supported.
dis_int may not be supported in the definition of the target.
If dis_int is supported, TOPPERS_SUPPORT_DIS_INT is defined as macro. If not supported, when dis_int is called, E_NOSPT error is returned or an error occurs when linking.

[Format]
ER errcd = dis_int(INTNO intno);

[Parameter]
INTNO intno　:　Interrupt number to be disabled

[Return value]
ER ercd　:　Error code
　E_OK　:　Completed normally
　E_CTX　:　Context error
　　・Called from non-task context
　E_NOSPT　:　Not supported error
　E_PAR　:　Parameter error
　　・intno is out of range.
　　・The interrupt request disabled flag of the target interrupt request line cannot be cleared because of the definition of the target.
　E_OBJ　: Object error
　　・The interrupt request line attribute is not configured to the target interrupt request line.

[Difference from μITRON 4.0 Specification]
intno, which was implementation-defined in μITRON 4.0 Specification, has been standardised.
In TOPPERS New Generation Kernel Integration Specification, it can be called while CPU is locked.

5.15.3. chg_ipm　Change Interrupt Priority Mask Level

[Function]
It changes the interrupt priority mask to the value specified with intpri.
intpri must be no less than TMINS_INTPRI and no more than TIPM_ENAALL. Otherwise, it results in E_PAR error. However, as an extension of the definition of the target, a smaller value than TMIN_INTPRI may be specified.
In case the interrupt priority mask is changed to TIPM_ENAALL, dispatch suspended state is released and a dispatch may take place. Also, a task exception handling routine may be initiated.
It specifies the interrupt priority mask level at the task execution to the value specified with ipm.
The specified value of IPM is inherited when other tasks are executed also.
This service call can be called only while CPU is unlocked in the task context.
If it is called from a non-task context or while CPU is unlocked, it returns E_CTX error.

[Format]
ER ercd = chg_ipm(PRI intpri);

[Parameter]
PRI intpri　　:　Interrupt priority mask level to be changed

[Return value]
ER ercd　Error code
E_OK　　:　Completed normally　　E_CTX　:　Context error
　　・Called from non-task context
　　・Called while CPU was locked
　E_PAR　:　Parameter error (Priority mask level specified is incorrect)

[Difference from μITRON 4.0 Specification]
μITRON 4.0 Specification, the name of the service call and the parameter name were implementation-defined service calls.

5.15.4. get_ipm Reference Interrupt Priority Mask Level

[Function]
It references the current value of the interrupt priority mask. The priority mask referenced is returned to the memory area specified with p_intpri.

[Format]
ER ercd = get_ipm(PRI *p_intpri);

[Parameter]
PRI *p_intpri : Pointer to the memory area in which the interrupt priority mask is placed.

[Return value]
ER ercd : Error code
E_OK : Completed normally
E_CTX : Context error
・Called from non-task context
・Called while CPU was locked

[Difference from µITRON 4.0 Specification]
µITRON 4.0 Specification, the name of the service call and the parameter name were implementation-defined service calls.

Contact:

Headquarters: Iijima Building, Nishi-gotanda 2-25-2, Shinagawa-ku, Tokyo 141-0031
TEL : 03-3493-7981　FAX : 03-3493-7993
Osaka Branch: 1205, Nishinakajima 6-2-3, Yodogawa-ku, Osaka 532-0011
TEL : 06-6304-5700　FAX : 06-6304-5705
Nagoya Branch: T&M Building 4-G, Sakae 5-19-3, Naka-ku, Nagoya 460-0008
TEL : 052-262-6451　FAX : 052-262-6460
E-mail : sales2@aicp.co.jp　URL : http://www.aicp.co.jp/