User's Manual

RI600/PX V.1.00

User's Manual

Real-time OS for RX Family MPU (Memory Protection Unit)

Renesas Electronics
www.renesas.com

Rev.1.00 Sep 2011

❑Preface

This manual describes how to use the RI600/PX, for the RX600 series micro-computer with MPU (Memory Protection Unit)before using the RI600/PX, please read this manual.

❑Notes on Descriptions

- Prefix
  Prefix "0x" indicates hexadecimal numbers. Numbers with no prefix are decimal.
- '\' is the directory delimiter.
- "cfg file" indicates kernel configuration file.
- XXX.YYY
  (1) Definition item of cfg file
  (2) Structure member
  (3) Bit(s) of register
- [XXX->YYY]
  "->" indicates menu option. (e.g.: [File -> Save])
- $(xxxx)
  Custom placeholder using in the High-performance Embedded Workshop

❑Trademarks

- μITRON is an acronym of the "Micro Industrial TRON" and TRON is an acronym of "The Real Time Operating system Nucleus". TRON, ITRON, and μITRON are the names of computer specifications and do not indicate a specific group of the commodity or the commodity.
  The μITRON4.0 specification is an open realtime-kernel specification defined by T-Engine Forum. The document of the μITRON4.0 specification can be downloaded from the T-Engine Forum homepage (http://www.t-engine.org/).
- Microsoft and Windows® are registered trademarks of Microsoft Corporation in the United States and/or other countries. The standard nomenclature of Windows is Microsoft Windows Operating System.
- All other product names are trademarks or registered trademarks of the respective holders.

❑Homepage

Various support information are available on the following Renesas Electronics homepage:

http://www.renesas.com/

❑Contents

1. Overview 1

1.1 Feature 1
1.2 Supplied Software Composition 2
1.3 Operating Environment 2

2. Introduction to the Kernel 3

2.1 Operating Principle of the Kernel 3
2.2 Service Call 4
2.3 Object 5
2.4 Task 6
2.4.1 Task State 6
2.4.2 Task Scheduling (Priority and Ready Queue) 8
2.4.3 Task Waiting Queue 9
2.5 System State 10
2.5.1 Task Context and Non-Task Context 10
2.5.2 Dispatching-Disabled State / Dispatching-Enabled State 10
2.5.3 CPU-Locked State / CPU-Unlocked State 11
2.5.4 Dispatching-Pending State 11
2.6 Processing Units and Precedence 12
2.7 Interrupts 13
2.7.1 Type of Interrupt 13
2.7.2 CPU Exception 13
2.7.3 I bit and IPL bit of the PSW Register 14
2.7.4 Disabling Interrupts 15
2.7.5 Usable Service Calls 15
2.7.6 Fast Interrupt Function of the RX Microcomputer 16
2.8 Stacks 16
2.9 Memory Protection 17
2.9.1 Overview 17
2.9.2 Domain, Memory Object, Access Permission Vector 17
2.9.3 Restriction in the Number of Memory Objects 18
2.9.4 Static Memory Object Registration Requirements 19
2.9.5 Trusted Domain 19
2.9.6 Changes Access Permission 20
2.9.7 Protection of User Stack 20
2.9.8 Checks Access Permission 20
2.9.9 Access Exception Handler 20
2.9.10 Processor Mode 20
2.9.11 Enables MPU 20
2.9.12 Area That Should Be the Inside of Memory Object 21
2.9.13 Area That Should Be the Outside of Memory Object 22
2.10 Kernel Idling 23
2.11 Task in Supervisor Mode 23

3. Kernel Functions 24

3.1 Module Structure 24
3.2 Module Overview 25
3.3 Task management Function 27
3.4 Task-Dependent Synchronization Function 30

3.5 Task Exception Handling Function 32
3.6 Semaphore 34
3.6.1 Functions....34
3.6.2 Priority Inversion Problem 36
3.7 Eventflag 37
3.8 Data Queue 39
3.9 Mailbox 41
3.9.1 Functions....41
3.9.2 Notes....43
3.10 Mutex 44
3.10.1 Functions....44
3.10.2 Base Priority and Current Priority 46
3.11 Message Buffer 47
3.12 Fixed-sized Memory Pool 49
3.12.1 Functions....49
3.12.2 Notes....50
3.13 Variable-Sized Memory Pool 51
3.13.1 Functions....51
3.13.2 About the Fragmentation of Free Spaces 53
3.13.3 Notes....53
3.14 Time Management Function 54
3.14.1 Task Timeout 54
3.14.2 Task Delay 54
3.14.3 Cyclic Handler 55
3.14.4 Alarm Handler....57
3.14.5 Accuracy of the Time 59
3.14.6 Precautions 60
3.15 System State Management Function 61
3.16 Interrupt Management Function 63
3.17 System Configuration Management Function 63
3.18 Object Reset Function 64
3.19 Memory Object Management Function 65

4. Data Macros 66

4.1 Data Types 66
4.2 Macro 67
4.2.1 Constant Macro 67
4.2.2 Function Macro....70

5. Service Call Reference.... 72

5.1 Header File....72
5.2 Service Call Return Values and Error Codes 72
5.2.1 Summary....72
5.2.2 Main Error Codes and Sub-Error Codes....72
5.3 System Status and Service Calls 72
5.3.1 Task Context and Non-Task Context....72
5.3.2 CPU-Locked State 73
5.3.3 Dispatching-Disabled State 73
5.3.4 Non-Kernel Interrupt Handler, etc. 73
5.4 Other Than µITRON Specification....73
5.5 Task Management Function 74
5.5.1 Creates Task (cre_tsk, acre_tsk) 75
5.5.2 Deletes Task (del_tsk)....78

5.5.3 Activates Task (act_tsk, iact_tsk)......79
5.5.4 Cancels Task Activation Request(can_act, ican_act)......80
5.5.5 Activates Task (with a Start Code) (sta_tsk, ista_tsk)......81
5.5.6 Terminates Invoking Task (ext_tsk)......82
5.5.7 Terminates and Deletes Invoking Task (exd_tsk)......83
5.5.8 Terminates Task (ter_tsk)......84
5.5.9 Changes Task Priority (chg_pri, ichg_pri)......85
5.5.10 Refers to Task Priority (get_pri, iget_pri)......86
5.5.11 Refers to Task State (ref_tsk, iref_tsk)......87
5.5.12 Refers to Task State (Simplified Version) (ref_tst, iref_tst)......89
5.6 Task Dependent Synchronization Function......90
5.6.1 Puts Task to Sleep (slp_tsk, tslp_tsk)......91
5.6.2 Wakeup Task (wup_tsk, iwup_tsk)......92
5.6.3 Cancels Task Wakeup Request (can_wup, ican_wup)......93
5.6.4 Forcibly Releases Task from WAITING State (rel_wai, irel_wai)......94
5.6.5 Suspends Task (sus_tsk, isus_tsk)......95
5.6.6 Resumes Suspended Task (rsm_tsk, irsm_tsk), Forcibly Resumes Suspended Task (frsm_tsk, ifrsm_tsk)......96
5.6.7 Delays Task (dly_tsk)......97
5.7 Task Exception Handling Function......98
5.7.1 Defines Task Exception Handling Routine (def_tex)......99
5.7.2 Raises Task Exception Handling (ras_tex, iras_tex)......101
5.7.3 Disables Task Exception (dis_tex)......103
5.7.4 Enables Task Exception (ena_tex)......104
5.7.5 Refers to Task Exception Disabled State (sns_tex)......105
5.7.6 Refers to Task Exception Handling State (ref_tex, iref_tex)......106
5.8 Synchronization and Communication Function (Semaphore)......107
5.8.1 Creates Semaphore (cre_sem, acre_sem)......108
5.8.2 Deletes Semaphore (del_sem)......110
5.8.3 Releases Semaphore Resource (sig_sem, isig_sem)......111
5.8.4 Acquires Semaphore Resource (wai_sem, pol_sem, ipol_sem, twai_sem)......112
5.8.5 Refers to Semaphore State (ref_sem, iref_sem)......113
5.9 Synchronization and Communication Function (Eventflag)......114
5.9.1 Creates Eventflag (cre_flg, acre_flg)......115
5.9.2 Deletes Eventflag (del_flg)......117
5.9.3 Sets Eventflag (set_flg, iset_flg)......118
5.9.4 Clears Eventflag (clr_flg, iclr_flg)......119
5.9.5 Waits for Eventflag (wai_flg, pol_flg, ipol_flg, twai_flg)......120
5.9.6 Refers to Eventflag State (ref_flg, iref_flg)......122
5.10 Synchronization and Communication Function (Data Queue)......123
5.10.1 Creates Data Queue (cre_dtq, acre_dtq)......124
5.10.2 Deletes Data Queue (del_dtq)......126
5.10.3 Sends to Data Queue (snd_dtq, psnd_dtq, ipsnd_dtq, tsnd_dtq, fsnd_dtq, ifsnd_dtq)......127
5.10.4 Receives from Data Queue (rcv_dtq, prcv_dtq, iprcv_dtq, trcv_dtq)......129
5.10.5 Refers to Data Queue State (ref_dtq, iref_dtq)......131
5.11 Synchronization and Communication Function (Mailbox)......132
5.11.1 Creates Mailbox (cre_mbx, acre_mbx)......133
5.11.2 Deletes Mailbox (del_mbx)......135
5.11.3 Sends to Mailbox (snd_mbx, isnd_mbx)......136
5.11.4 Receives from Mailbox (rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx)......138
5.11.5 Refers to Mailbox State (ref_mbx, iref_mbx)......140
5.12 Extended Synchronization and Communication Function (Mutex)......141
5.12.1 Creates Mutex (cre_mtx, acre_mtx)......142

5.12.2 Deletes Mutex (del_mtx) 144
5.12.3 Locks Mutex (loc_mtx, ploc_mtx, tloc_mtx) 145
5.12.4 Unlocks Mutex (unl_mtx) 146
5.12.5 Refers to Mutex State (ref_mtx) 147
5.13 Extended Synchronization and Communication Function (Message Buffer) 148
5.13.1 Creates Message Buffer (cre_mbf, acre_mbf) 149
5.13.2 Deletes Message Buffer (del_mbf) 151
5.13.3 Sends to Message Buffer (snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf) 152
5.13.4 Receives from Message Buffer (rcv_mbf, prcv_mbf, trcv_mbf) 154
5.13.5 Refers to Message Buffer State (ref_mbf, iref_mbf) 156
5.14 Memory Pool management Function (Fixed-sized Memory Pool) 157
5.14.1 Creates Fixed-sized Memory Pool (cre_mpf, acre_mpf) 158
5.14.2 Deletes Fixed-sized Memory Pool (del_mpf) 160
5.14.3 Acquires fixed-sized memory block (get_mpf, pget_mpf, ipget_mpf, tget_mpf) 161
5.14.4 Releases Fixed-sized Memory Pool (rel_mpf, irel_mpf) 163
5.14.5 Refers to Fixed-sized Memory Pool State (ref_mpf, iref_mpf) 164
5.15 Memory Pool management Function (Variable-sized Memory Pool) 165
5.15.1 Creates Variable-sized Memory Pool (cre_mpl, acre_mpl) 166
5.15.2 Deletes Variable-sized Memory Pool (del_mpl) 169
5.15.3 Acquires variable-sized Memory Block (get_mpl, tget_mpl, pget_mpl, ipget_mpl) 170
5.15.4 Release Variable-sized Memory Block (rel_mpl) 172
5.15.5 Refers to Variable-sized Memory Pool State (ref_mpl, iref_mpl) 173
5.16 Time Management Function (System Time) 174
5.16.1 Sets System Time (set_tim, iset_tim) 175
5.16.2 Refer to System Time (get_tim, iget_tim) 176
5.16.3 Supplies Time Tick (isig_tim) 177
5.17 Time Management Function (Cyclic Handler) 178
5.17.1 Creates Cyclic Handler (cre_cyc, acre_cyc) 179
5.17.2 Deletes Cyclic Handler (del_cyc) 181
5.17.3 Starts Cyclic Handler Operation (sta_cyc, ista_cyc) 182
5.17.4 Stops Cyclic Handler Operation (stp_cyc, istp_cyc) 183
5.17.5 Refers to Cyclic Handler State (ref_cyc, iref_cyc) 184
5.18 Time Management Function (Alarm Handler) 185
5.18.1 Creates Alarm Handler (cre_alm, acre_alm) 186
5.18.2 Deletes Alarm Handler (del_alm) 188
5.18.3 Starts Alarm Handler Operation (sta_alm, ista_alm) 189
5.18.4 Stops Alarm Handler (stp_alm, istp_alm) 190
5.18.5 Refers to Alarm Handler State (ref_alm, iref_alm) 191
5.19 System State Management Function 192
5.19.1 Rotates Task Precedence (rot_rdq, irot_rdq) 193
5.19.2 Refers to Task ID in the RUNNING State (get_tid, iget_tid) 194
5.19.3 Locks the CPU (loc_cpu, iloc_cpu) 195
5.19.4 Unlocks the CPU (unl_cpu, iunl_cpu) 197
5.19.5 Disables Dispatching (dis_dsp) 198
5.19.6 Enables Dispatching (ena_dsp) 199
5.19.7 Refers to Context State (sns_ctx) 200
5.19.8 Refers to CPU-Locked State (sns_loc) 201
5.19.9 Refers to Dispatching-Disabled Dtate (sns_dsp) 202
5.19.10 Refers to Dispatching-Pending State (sns_dpn) 203
5.19.11 Starts Kernel (vsta_knl, ivsta_knl) 204
5.19.12 Terminates System (vsys_dwn, ivsys_dwn) 205
5.20 Interrupt management Function 206
5.20.1 Changes Interrupt Mask (chg_ims, ichg_ims) 207

5.20.2 Refers to Interrupt Mask (get_ims, iget_ims)..........208
5.20.3 Returns from kernel interrupt handler (ret_int)..........209
5.21 System Configuration Management Function..........210
5.21.1 Refers to Version Information (ref_ver, iref_ver)..........211
5.22 Object Reset Function..........213
5.22.1 Resets Data Queue (vrst_dtq)..........214
5.22.2 Resets Mailbox(vrst_mbx)..........215
5.22.3 Resets Message Buffer (vrst_mbf)..........216
5.22.4 Resets Fixed-sized Memory Pool (vrst_mpf)..........217
5.22.5 Resets Variable-sized Memory Pool (vrst_mpl)..........218
5.23 Memory Object Management Function..........219
5.23.1 Registers Memory Object (ata_mem)..........220
5.23.2 Unregisters Memory Object (det_mem)..........222
5.23.3 Changes Access Permission Vector for Memory Object (sac_mem)..........223
5.23.4 Checks Access Permission (vprb_mem)..........224
5.23.5 Refers to Memory Object State (ref_mem)..........225

6. How to Write Application..........226

6.1 Header Files..........226
6.2 Handling of Variables..........226
6.3 Task..........227
6.3.1 Coding..........227
6.3.2 CPU State at Start..........228
6.4 Task Exception Handling Routine..........229
6.4.1 Coding..........229
6.4.2 CPU State at Start..........229
6.5 Interrupt Handler..........230
6.5.1 Coding..........230
6.5.2 CPU State at Start..........230
6.6 Time Event Handler (Cyclic Handler and Alarm Handler)..........231
6.6.1 Coding..........231
6.6.2 CPU State at Start..........231
6.7 Access Exception Handler..........232
6.7.1 Summary..........232
6.7.2 Coding..........232
6.7.3 CPU State at Start..........233
6.8 System-Down Routine..........234
6.8.1 Summary..........234
6.8.2 Coding..........234
6.8.3 CPU State at Start..........236
6.9 Precautions to Take when Using Floating-Point Arithmetic Instructions..........237
6.10 Precautions to Take when Using a Microcomputer that Supports the DSP Function..........238

7. Procedure for Generating the Load Module..........240

7.1 Summary..........240
7.2 Creating Startup File (resetprg.c)..........242
7.3 Kernel Libraries..........247
7.4 Section List..........247
7.4.1 Naming Convention of Section..........247
7.4.2 List of RI600/PX Sections..........248
7.4.3 "aligned_section" Option for Linker..........248
7.4.4 Attention Concerning L and W section..........249
7.5 Service Call Information File (mrc file) and Essential Compiler Option..........249

7.6 Notes 250
7.6.1 Processor mode 250
7.6.2 Address 0....250

8. Configurator (cfg600px)....251

8.1 Creating a Configuration File (cfg File) 251
8.2 Representation Format in cfg File....251
8.3 Default cfg File....253
8.4 Definition Items in cfg File 253
8.4.1 System Definition (system)....254
8.4.2 Precautions to Take when Defining system.context 256
8.4.3 System Clock Definition (clock)....258
8.4.4 Maximum ID Definition (maxdefine[])....260
8.4.5 Domain Definition (domain[])....264
8.4.6 Memory Object Definition (memory_object[])....265
8.4.7 Task Definition (task[])....266
8.4.8 Semaphore Definition (semaphore[]) 269
8.4.9 Eventflag Definition (flag[])....270
8.4.10 Data Queue Definition (dataqueue[])....272
8.4.11 Mailbox Definition (mailbox[])....273
8.4.12 Mutex Definition (mutex[]) 274
8.4.13 Message Buffer Definition (message_buffer[]) 275
8.4.14 Fixed-sized Memory Pool (memorypool[])....276
8.4.15 Variable-sized Memory Pool Definition (variable_memorypool[])....278
8.4.16 Cyclic Handler Definition (cyclic_hand[]) 280
8.4.17 Alarm Handler Definition (alarm_hand[])....282
8.4.18 Relocatable Vector Definition (interrupt_vector[])....283
8.4.19 Fixed Vector Definition (interrupt_fvector[])....285
8.5 Executing the Configurator....287
8.5.1 Outline of the Configurator 287
8.5.2 Environment Settings 288
8.5.3 Configurator Start Procedure....288
8.5.4 Command Options....288
8.6 Errors Messages....289
8.6.1 Error Output Format and Error Levels....289
8.6.2 List of Messages....289

9. Table Generation Utility (mkritblpx)....292

9.1 Summary 292
9.2 Environment Setup....293
9.3 Table Generation Utility Start Procedure 293
9.4 Notes....293

10. Sample Program....294

10.1 Summary of Sample Program 294
10.2 Generating a RI600/PX Project....296
10.3 Generated Files 296
10.4 Memory Map....297
10.4.1 RAM Area....297
10.4.2 ROM Area 297
10.4.3 Memory Objects....298
10.4.4 User Stacks....300
10.5 Setting of Build Tools concerning Sections 301

10.5.1 Standard Library Generator 301
10.5.2 C/C++ Compiler 301
10.5.3 Linker 301
10.6 Example of Dealing with Access Violation 301

11. Stack Size Estimation 302

11.1 Types of Stacks 302
11.2 "Call Walker" 302
11.3 User Stack Size Estimation 303
11.4 System Stack Size Estimation 304

RI600/PX V.1.00

Real-time OS for RX Family with MPU (Memory Protection Unit)

R20UT0606EJ0100
Rev.1.00
Sep 01, 2011

1. Overview

1.1 Feature

(1) Compliant with μITRON 4.0 Specification

The RI600/PX was developed based on the RI600/4, which conforms to the latest version of μITRON 4.0 specification and with additional memory protection function of the μITRON 4.0 protection extension. Therefore, the knowledge acquired from μITRON specification-related various publications or seminars, etc. can be made use of directly. In addition, the application programs developed using other μITRON-compliant real-time OS's can be ported to the RI600/PX relatively easily.

(2) Increased processing speed

By taking advantage of the RX600-series microcomputer architecture, the processing speed is significantly increased.

(3) Systems always built to minimum size by automatically selecting only the necessary modules

The RI600/PX kernel is supplied in RX600-series object library form. This means that from among numerous functional modules of the kernel, only those that an application uses are automatically selected, thanks to the functionality the linkage editor has. Therefore, systems are always generated in minimum size.

(4) Efficient development possible by making use of the integrated development environment

Renesas' integrated development environment, or High-performance Embedded Workshop, can be used as you proceed with development work. High-performance Embedded Workshop supports the function to generate workspaces for RI600/PX-compatible applications. Furthermore, the real-time OS debug functions of High-performance Embedded Workshop are also usable for the RI600/PX.

(5) Kernel building made easy by the configurator

The RI600/PX comes with the command line configurator cfg600px. By only writing various kernel-building information in a text-format cfg file, it is possible to build the kernel. This configurator information, as it is created in text form, can be altered and maintained easily.

1.2 Supplied Software Composition

(1) Kernel

This is the real-time OS body.

(2) cfg600px (command line configurator)

This is a tool to configure the kernel. It accepts as its input the cfg file created by the user and outputs a kernel definition file.

(3) mkritblpx (table generation utility)

This is the command line tool that by gathering the service call information used by application, generates the service call and interrupt vector tables most suitable for the application.

1.3 Operating Environment

Table 1.1 Operating Environment

Item	Operating Environment
Target CPU core	RX600 series with MPU (Memory Protection Unit)
Host machine	IBM-PC/AT compatible machine operated under Windows® XP, Windows Vista® or Windows® 7
Compiler	Renesas C/C++ Compiler Package for RX Family V.1.01 or later

2. Introduction to the Kernel

2.1 Operating Principle of the Kernel

The kernel program is the nucleus of the realtime operating system. The kernel enables one CPU to appear as if multiple CPUs are operating. How does the kernel do this? As is shown in Figure 2.1, the kernel switches operation between various tasks as required.

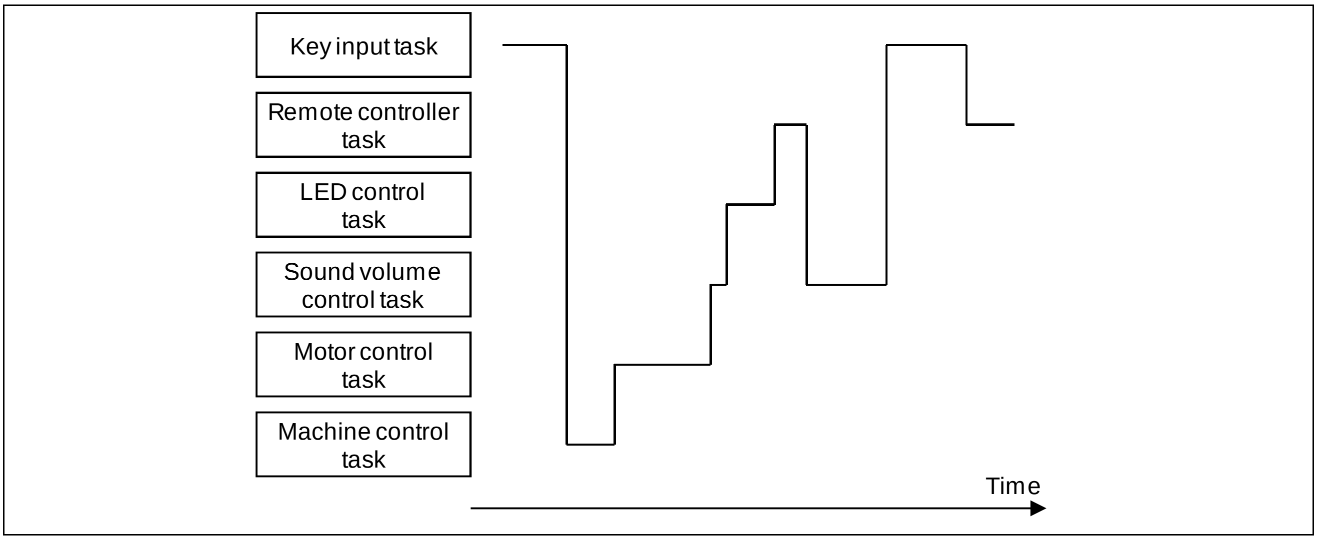

Figure 2.1 Operation of Multiple Tasks

This switching between tasks is called task dispatch. The kernel dispatches tasks in the following cases.

- When a task itself requests a dispatch
- When an event (such as an interrupt) outside the current task requests a dispatch

This means that tasks are not switched at predetermined intervals as in a time-sharing system. This type of scheduling is generally called event-driven.

After a task is dispatched, execution of the task resumes from the point at which it was previously suspended (Figure 2.2).

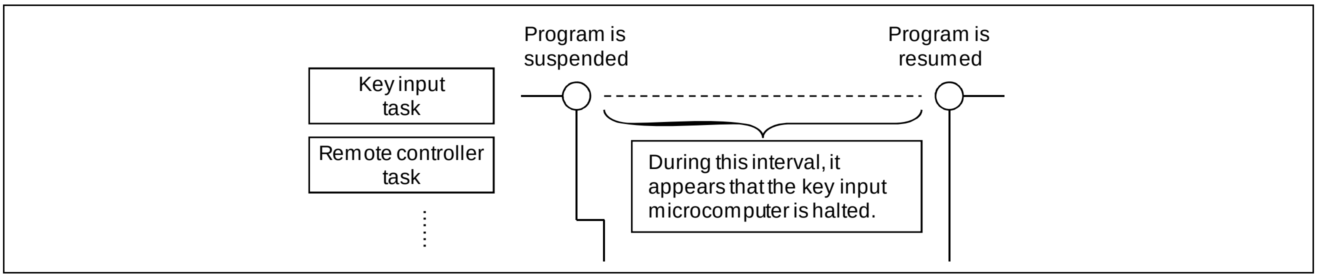

Figure 2.2 Suspending and Resuming a Task

In Figure 2.2, it appears to the programmer that the key input task or its microcomputer is halted while another task assumes execution control.

By restoring the contents of CPU registers that were stored when a task was suspended, the kernel resumes the execution of a task from the state in which it was suspended. In other words, dispatching a task means saving the contents of the CPU registers for the task currently being executed in a memory area prepared for the management of that task, and restoring the contents of the CPU registers for the task for which execution is being resumed (Figure 2.3)

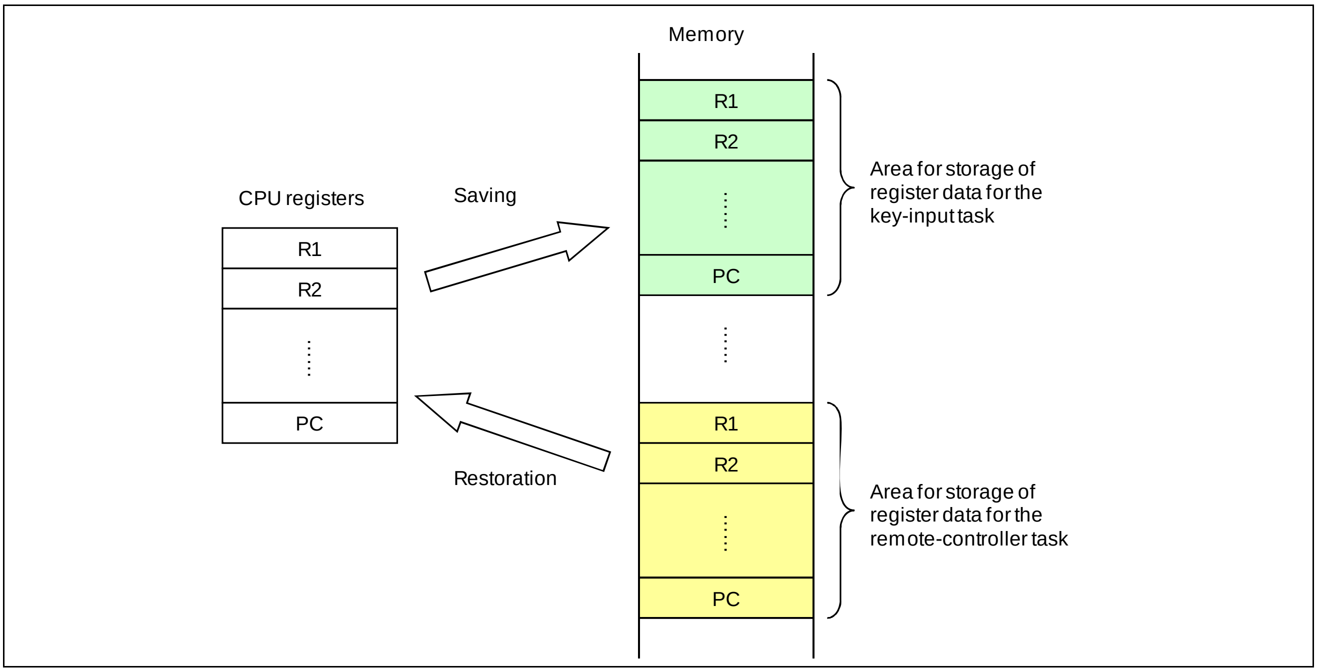

Figure 2.3 Task Dispatch

As well as the CPU registers, task execution requires stack areas. Separate stack area must be allocated for each task.

2.2 Service Call

How does the programmer use the kernel functions in a program?

First, it is necessary to call up kernel function from the program in some way or other. Calling a kernel function is referred to as a service call. Task activation and other processing operations can be initiated by such a service call (Figure 2.4).

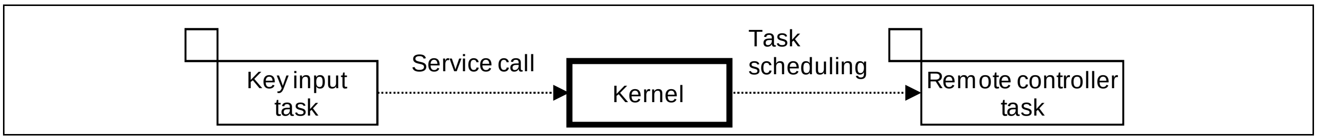

Figure 2.4 Service Call

This service call is realized by a function call when the application program is written in C language, as shown below

```
act_tsk(ID_TASK1);
```

2.3 Object

The processing objectives of service calls, such as tasks or semaphores, are called objects.

The operation to recognize objects to the kernel is called "create". The object can be created by cfg file statically or by using service call (cre_??? or acre_???) dynamically.

Objects are distinguished by their ID numbers. However, if a task number is directly written in a program, the resultant program would be very low in readability. If, for instance, the following is entered in a program, the programmer is constantly required to know what the No. 1 task is.

```
act_tsk(1);
```

Further, if this program is viewed by another person, he/she does not understand at a glance what the No. 1 task is.

To avoid such inconvenience, the RI600/PX provides means of specifying the task, which is created by cfg file, by name (ID name). The program named "configurator (cfg600px)", which is supplied with the RI600/PX, then automatically converts the task name to the task ID number. To be more specific, the cfg600px outputs the header file "kernel_id.h" which contains definitions of the following type, associating task ID names with task ID numbers.

```
#define ID_TASK1 1
```

Figure 2.5 is a schematic view of the task identification system.

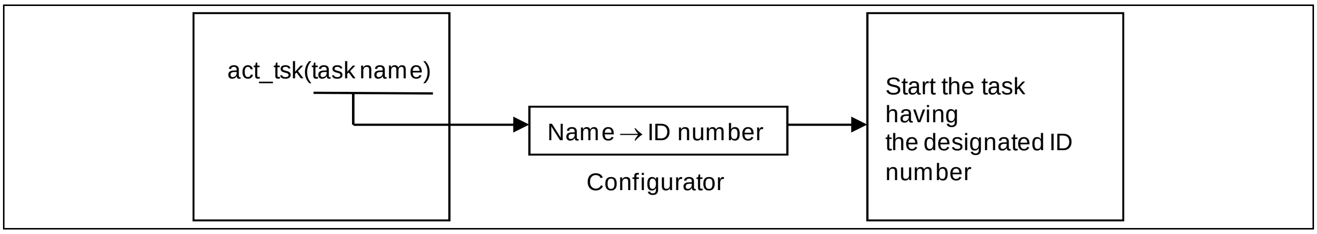

Figure 2.5 Task Identification

With this task identification system, our earlier example is now as follows.

```
act_tsk(ID_TASK1);  /* Start the task having the ID name "ID_TASK" */
```

This call specifies invocation of the task corresponding to "ID_TASK1". Also note that the compiler's pre-processor converts task names to ID numbers in the generation of an executable program. Therefore, this feature does not reduce processing speeds.

Although the example on this section just referred to task identification, other objects that have ID numbers can also be given ID names.

2.4 Task

2.4.1 Task State

The kernel checks the task state to control whether to execute a task. For example, Figure 2.6 shows the state of the key input task and its execution control. When a key input is detected, the kernel must execute the key input task; that is, the key input task enters the RUNNING state. While waiting for a key input, the kernel does not need to execute the key input task; that is, the key input task is in the WAITING state.

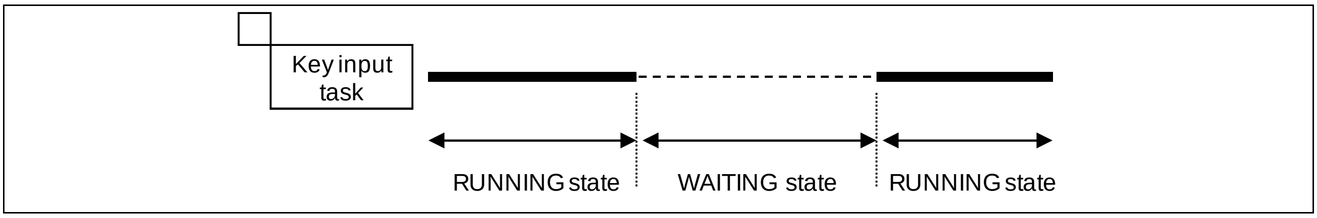

Figure 2.6 Task States

The kernel controls transitions between six states, including the RUNNING and WAITING states, as shown in Figure 2.7. A task makes the transitions between these six states.

(1) NON-EXISTENT state

The task has not been registered in the kernel. This is a virtual state where by task resource such as user stack has not been reserved and initialized.

(2) DORMANT state

The task has been registered in the kernel, but has not yet been initiated, or has already been terminated.

(3) READY state

The task is ready for execution, but cannot be executed because another higher priority task is currently running.

(4) RUNNING state

The task is currently running. The kernel puts the READY task with the highest priority in the RUNNING state.

(5) WAITING state

When the task issues a service call such as tslp_tsk and the specified conditions are not satisfied, the task enters the WAITING state. A task is released from the WAITING state by the service call (such as wup_tsk) that corresponds to the call which initiated the WAITING state, after which the task enters the READY state.

(6) SUSPENDED state

A task has been suspended by another task through sus_tsk.

(7) WAITING-SUSPENDED state

This state is a combination of the WAITING state and SUSPENDED state.

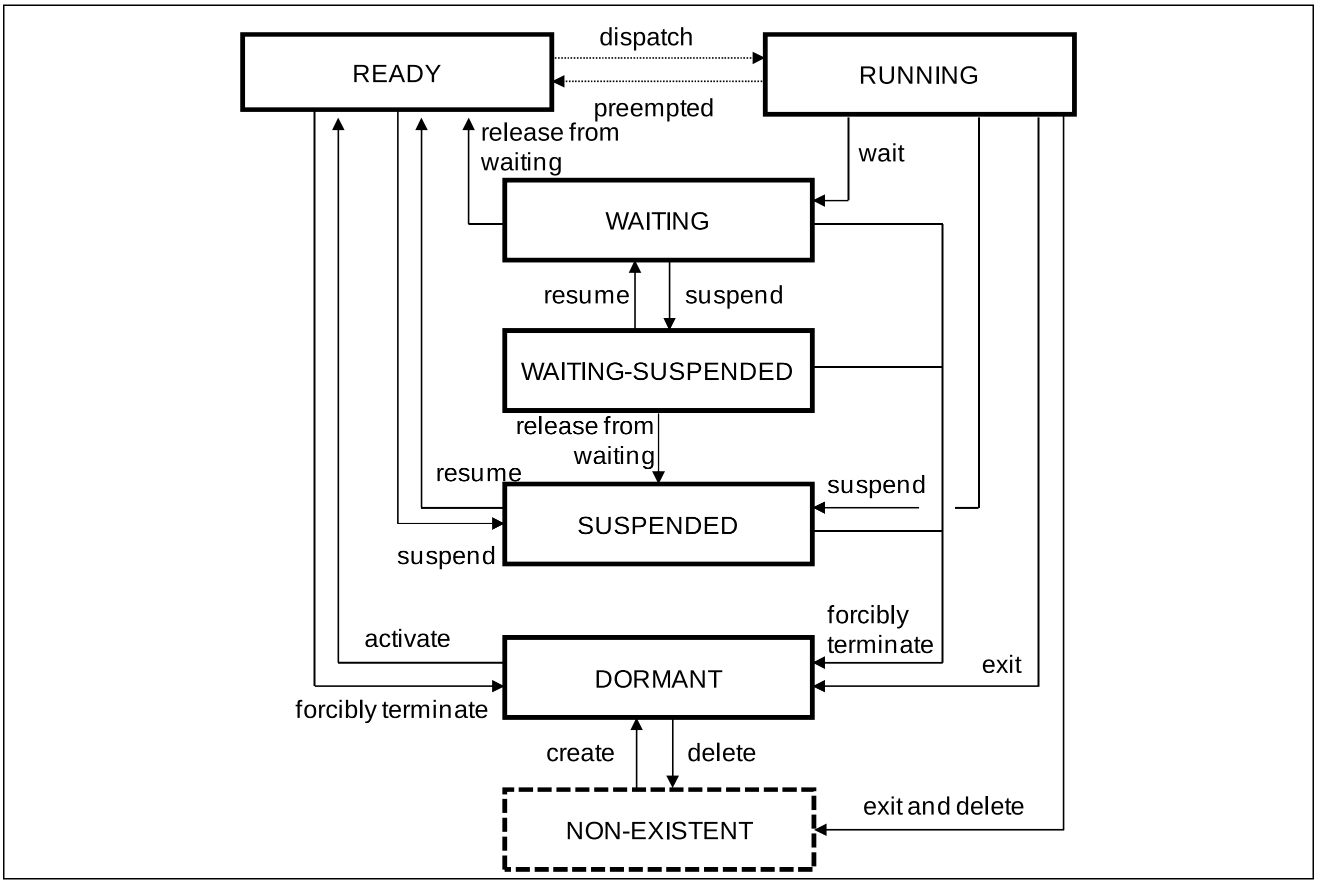

Figure 2.7 Task State Transition Diagram

2.4.2 Task Scheduling (Priority and Ready Queue)

For each task, a task priority is assigned to determine the priority of processing. A smaller value indicates a higher priority level and level 1 is the highest priority. The range of available priorities is 1 to system.priority as defined in the cfg file.

The kernel selects the highest-priority task from among the READY tasks and puts it in the RUNNING state.

The same priority can be assigned to multiple tasks. When there are multiple READY tasks with the highest priority, the kernel selects the first task to have become READY and puts it in the RUNNING state. To implement this behavior, the kernel has ready queues, which are queues of READY task waiting for execution.

Figure 2.8 shows the ready queue configuration. A ready queue is provided for each priority level, and the kernel selects the task at the head of the non-empty ready queue for the highest priority and puts it in the RUNNING state.

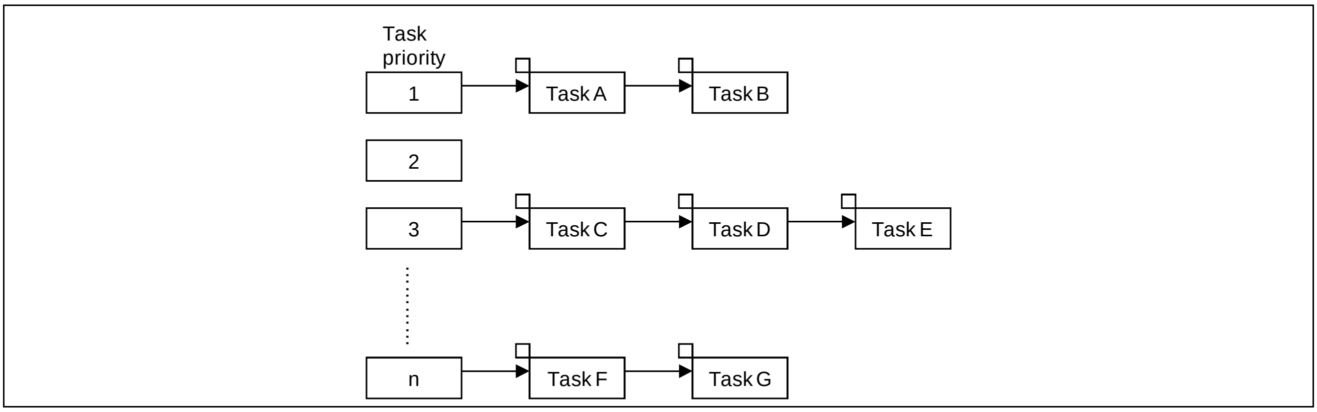

Figure 2.8 Ready Queues (Waiting for Execution)

2.4.3 Task Waiting Queue

A service call can make a task wait (enter the WAITING state) until a condition designated in terms of objects (such as semaphores and eventflags) has been satisfied. For some types of objects, two or more tasks may be in the WAITING state. Attributes that select the order in which waiting tasks are handled are specifiable when the objects are created. The specifiable attributes are TA_TFIFO (handling on an FIFO basis) or TA_TPRI (handling on a priority basis).

Tasks leave the WAITING state in the order specified for the waiting queue. Figure 2.9 and Figure 2.10 show the order of task handling for objects with the respective attributes, where task D (priority: 9), task C (priority: 6), task A (priority: 1), and task B (priority: 5) have joined the waiting queue, in that order.

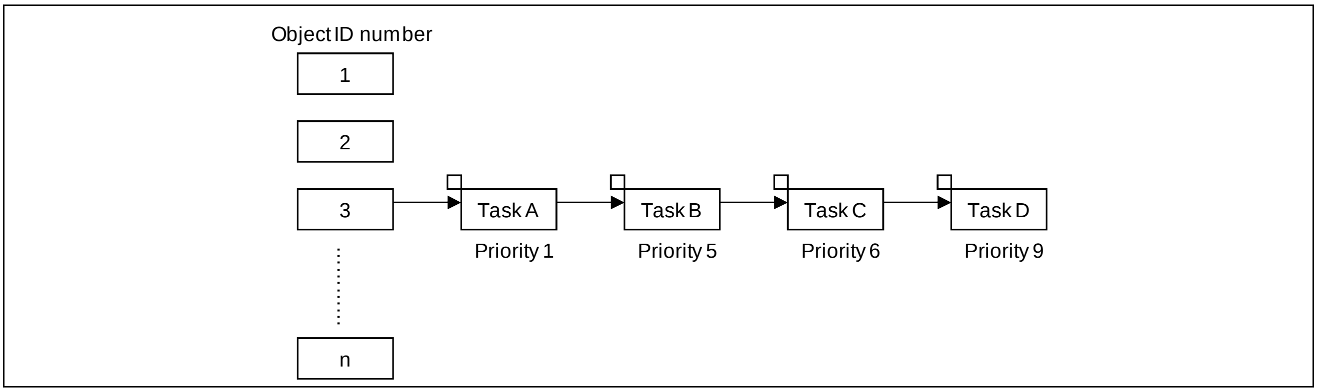

Figure 2.9 Waiting Queue with the Attribute TA_TPRI

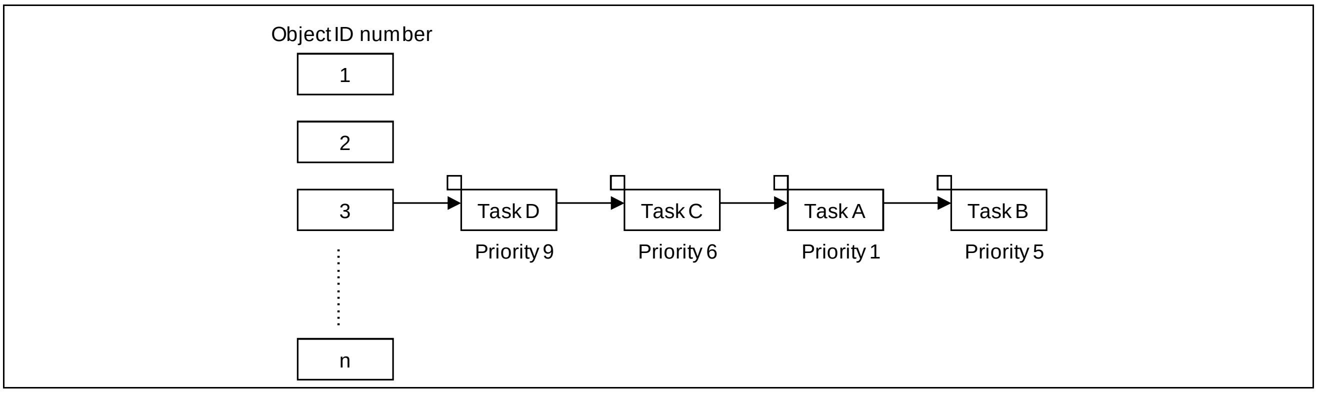

Figure 2.10 Waiting Queue with the Attribute TA_TFIFO

2.5 System State

The system state is classified into the following orthogonal states.

- Task context / non-task context
- Dispatching-disabled state / Dispatching-enabled state
- CPU-locked state / CPU-unlocked state

The system operations and available service calls are determined based on the above system states.

2.5.1 Task Context and Non-Task Context

System is in either task contexts or non-task contexts. The difference between task contexts and non-task contexts is described in Table 2.1.

Table 2.1 Task Context and Non-Task Context

Item	Task Context	Non-Task Context
Available service calls	Service calls that can be called from task context	Service calls that can be called from non-task context
Task scheduling	Refer to sections 2.5.2 and 2.5.3	Does not occur

The following forms of processing are executed in non-task contexts.

- Interrupt handlers
- Time event handlers (cyclic handlers and alarm handlers)
- Access exception handler

2.5.2 Dispatching-Disabled State / Dispatching-Enabled State

System is in either the dispatching-disabled state or the dispatching-enabled state. In the dispatching-disabled state, task scheduling is not allowed and service calls that place the current task in the WAITING state cannot be used.

The following service calls changes the system state to the dispatching-disabled state.

- dis_dsp
- chg_ims, that changes the interrupt mask to other than 0

The following service calls changes the system state to the dispatching-enabled state.

- ena_dsp
- ext_tsk
- exd_tsk
- chg_ims, that changes the interrupt mask to 0

Issuing the sns_dsp service call will check whether the system is in the dispatching-disabled state or not.

It is possible to control it exclusively with tasks by using the dispatching-disabled state. However, because other tasks cannot executes while the dispatching-disabled, it influences the behavior of the entire system. To control between tasks exclusively, semaphore or mutex is recommended to be used as much as possible. It is necessary to shorten the time of the dispatching-disabled state as much as possible.

2.5.3 CPU-Locked State / CPU-Unlocked State

System is in either the CPU-locked state or the CPU-unlocked state. In the CPU-locked state, interrupts and task scheduling are not allowed. Note, however, that non-kernel interrupts are allowed.

Any service calls that make tasks enter the WAITING state cannot be issued.

Issuing the loc_cpu or iloc_cpu service call changes the system state to the CPU-locked state. Issuing an unl_cpu, iunl_cpu, ext_tsk or exd_tsk will return the system state to the CPU-unlocked state. In addition, issuing the sns_loc service call will check whether the system is in the CPU-locked state or not.

Service calls that can be issued in the CPU-locked state are restricted to those listed in Table 2.2.

Table 2.2 Service Calls that can be issued in the CPU-Locked State

loc_cpu, iloc_cpu	unl_cpu, iunl_cpu	sns_ctx	sns_loc	sns_dsp
sns_dpn	vsta_knl, ivsta_knl	vsys_dwn, ivsys_dwn	ext_tsk * exd_tsk *	sns_tex

Note: These service call cancels the CPU-locked state.

It is possible to control it exclusively with kernel interrupt processings or tasks and kernel interrupt processings by using the CPU-locked state. However, because other tasks cannot execute and kernel interrupts are masked while the CPU-locked, it influences the behavior of the entire system. It is necessary to shorten the time of the CPU-locked state as much as possible.

2.5.4 Dispatching-Pending State

The state that task-dispatching is pended is called the dispatching-pending state. To be more specific, each of the following cases corresponds to the dispatch-pending state.

- Non-task context
- Dispatching-disabled state
- CPU-locked state

The sns_dpn service call can be used to check if the system is in the dispatch-pending state.

2.6 Processing Units and Precedence

An application program is executed in the following processing units.

(1) Task
A task is a unit controlled by multitasking.

(2) Interrupt handler
An interrupt handler is executed when an interrupt occurs.

(3) Time Event Handler (Cyclic Handler and Alarm Handler)
A time event handler is executed when a specified cycle or time has been reached.

(4) Access exception handler
The access exception handler is executed when a task or task exception handling routine accesses to the memory area that has not been permitted.

The various processing units are processed in the following order of precedence.

(1) Interrupt handlers, time event handlers

(2) Access exception handler

(3) Dispatcher (part of kernel processing)

(4) Task

The dispatcher is kernel processing that switches the task being executed. Since interrupt handlers and time event handlers have higher precedence than the dispatcher, no tasks are executed while these handlers are executing.

The precedence of an interrupt handler becomes higher when the interrupt level is higher.

The precedence of a time event handler is the same as the timer interrupt level (clock.IPL).

The order of precedence for tasks depends on the priority of the tasks.

2.7 Interrupts

When an interrupt occurs, the interrupt handler defined in the cfg file is initiated.

2.7.1 Type of Interrupt

Interrupt is classified into kernel interrupt and non-kernel interrupt.

- **Kernel Interrupt**

  An interrupt whose interrupt priority level is lower than or equal to the kernel interrupt mask level (system.system_IPL) is called the kernel interrupt.
  Service calls can be issued from within a kernel interrupt handler.
  Note, however, that handling of kernel interrupts generated during kernel processing may be delayed until the interrupts become acceptable.

- **Non-Kernel Interrupt**

  An interrupt whose interrupt priority level is higher than the kernel interrupt mask level (system.system_IPL) is called the non-kernel interrupt. The non-maskable interrupt (NMI) is handled as non-kernel interrupt.
  No service calls can be issued from within a non-kernel interrupt handler.
  Non-kernel interrupts generated during service-call processing are immediately accepted whether or not kernel processing is in progress

Figure 2.11 shows the relationship between the non-kernel interrupt handlers and kernel interrupt handlers where the kernel mask level is set to 10.

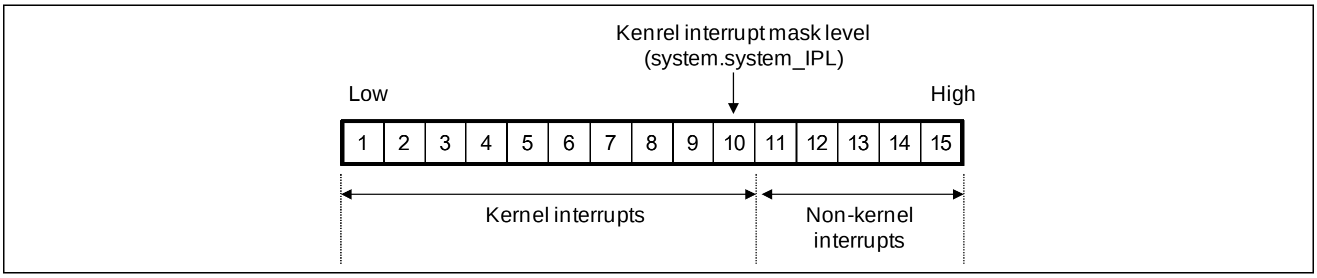

Figure 2.11 Interrupt Handler IPLs

2.7.2 CPU Exception

The following CPU exceptions are handled as non-kernel interrupt.

1. Unconditional Trap (INT, BRK instruction) (INT #1 to #8 are reserved by kernel)
2. Undefined instruction exception
3. Privileged instruction exception
4. Floating-point exception

On the other hand, access exception is handled as kernel interrupt.

2.7.3 I bit and IPL bit of the PSW Register

In the CPU specification, all interrupts are masked when I bit is 0, and interrupts at the level below the IPL bit value is masked when I bit is 1.

Please refer to the next paragraph for the change in I bit and the IPL bit by the application.

(1) Task, task exception handling routine

An initial value of the I bit is 1, and an initial value of the IPL bit is 0. In a word, all the interruptions are permitted.

In the change of IPL, there is a method of using loc_cpu or chg_ims.

An task must not change the I bit to 0. Note, the RI600/PX does not provide a method to change the I bit.

(2) Interrupt Handler

An initial value of the I bit is 1 when the handler is executed to be enable nested interrupts (pragma_switch=E). When it is not so the case (pragma_switch=E is not defined), an initial value of the I bit is 0 and no nested interrupt is enabled.

An initial value of the IPL bit when interrupt handler is started is as follows.

- Interrupt : The interrupt priority level
- CPU exception : Same level before exception

To change the IPL level, there is a method of using iloc_cpu or ichg_ims.

In the change of IPL, there is a method of using iloc_cpu or ichg_ims.

An interrupt handler can also change the I bit and IPL bit directly.

(3) Time Event Handler

An initial value of the I bit is 1

An initial value of the IPL bit is the interrupt priority level of the timer interrupt.(clock.IPL in the .cfg file.). In the change of IPL, there is a method of using iloc_cpu or ichg_ims.

A time event handler can also change the I bit and IPL bit directly.

(4) Contorl in a service call

(a) I bit

When a service call is called from the task context, the service call runs while switching the I bit to 0 and 1.
When a service call is called from the non-task context, the I bit is maintained in the service call.

(b) IPL bit

A service call runs while switching the IPL bit to the value before calling and the kernel interrupt mask level (system.system_IPL).

(c) State after a service call ends

The PSW register return to the value before calling. However, the IPL bit is changed by chg_ims, ichg_ims, loc_cpu, iloc_cpu, unl_cpu, or iunl_cpu.

2.7.4 Disabling Interrupts

To disable interrupts, follow one of the methods described below.

(1) Disable unspecified interrupts (Change I bit and IPL bit of the PSW register)

(a) Putting into the CPU-locked state
In the CPU-locked state, PSW.IPL is set to the kernel interrupt mask level (system.system_IPL). Therefore, it is only the kernel interrupts that are disabled in the CPU-locked state. To disable non-kernel interrupts, follow the method (b) or (c).
Also, when in the CPU-locked state, the usable service calls are subject to some limitations. See Section 2.5.3, "CPU-Locked State / CPU-Unlocked State."

(b) Use chg_ims or ichg_ims to alter PSW.IPL.
In non-task context, do not lower the IPL bit more than the value when the handler is started.
In task-context, the system enters to dispatching-disabled state when the IPL bit is changed to other than 0, and the system enters to dispatching-enabled state when the IPL bit is changed to 0.
Usually, do not call ena_dsp while having changed the IPL bit to other than 0 If calling ena_dsp, the system enters to dispatching-enabled state. The PSW register is changed to the value for dispatched task, so the IPL may be lowered.

(c) Handlers (non-task context) can change the I bit and IPL bit directly. However, do not lower the IPL bit more than the value when the handler is started.
Note, the compiler provides following intrinsic functions for handling PSW register.

- set_ipl() : Changes the IPL bit of the PSW register
- get_ipl() : Refers to the IPL bit of the PSW register
- set_psw() : Changes the PSW register
- get_psw() : Refers to the PSW register

Note, PSW.I must not be cleared during executing in the task-conetxt.

(2) Disable specified interrupts

Change Interrupt Request Registers in the interrupt controller (ICU) or control registers of the I/O to disable specified interrupts.

2.7.5 Usable Service Calls

Since the interrupt hander is classified as belonging to non-task contexts, the task context-only service calls cannot be used in the interrupt handler.

When the PSW.IPL is larger than or equal to the kernel interrupt mask level (system.system_IPL), such as non-kernel interrupt handlers, do not call service calls[1], If calling, the service call returns error E_CTX. In this case, the PSW.IPL temporarily falls on the kernel interrupt mask level. Please use this error only for the purpose of debugging.

[1] Except chg_ims, ichg_ims, get_ims, iget_ims, vsta_knl, ivsta_knl, vsys_dwn, and ivsys_dwn

2.7.6 Fast Interrupt Function of the RX Microcomputer

The RX microcomputer supports the "fast interrupt" function. Only one interrupt source can be made the fast interrupt. The fast interrupt is handled as the one that has interrupt priority level 15. To use the fast interrupt function, make sure there is only one interrupt source that is assigned interrupt priority level 15.

For the fast interrupt function to be used in this kernel, it is necessary that the interrupt concerned be handled as an non-kernel interrupt. In other words, the kernel interrupt mask level (system.system_IPL) must be set to 14 or below.

And "os_int = NO;" and "pragma_switch = F;" are required for interrupt_vector[] definition.

And the FINTV register must be initialized to the start address of the handler by application.

2.8 Stacks

Stack is classified into user stack and system stack.

- **User Stack**

  One user stack is provided for each task.
  When a task is created statically by using cfg file, the stack size and section name are specified.
  When a task is created dynamically by using cre_tsk or acre_tsk service call, the stack size and start address are specified..

- **System Stack**

  The system stack is used by non-task context and the kernel. There is one system stack in the system.
  The system stack is generated by specifying system.stack_size in the cfg file.

2.9 Memory Protection

2.9.1 Overview

The kernel achieves following memory access protection function by using MPU (Memory Protection Unit) found in MCU. Note, handlers can access all address space.

(1) Detection of illegal access by tasks and task exception handling routines

Tasks and task exception handling routines can access only permitted memory objects. The access exception handler will be initiated if a task or task exception handling routine access the area that has not been permitted.

(2) Protection of user stack

The user stack of each task is inaccessible from other tasks. The access exception handler will be initiated if an user stack overflows or a task accesses an user stack for another task.

(3) Detection of illegal access by the kernel

Some service calls receives pointers as argument. The kernel inspects whether the invoking task can access to the area indicated by the pointer. The service call returns E_MACV error when the invoking task does not have the access permission to the area.

And some service calls saves the context registers of the invoking task. If the user stack will overflow at the time, the service call returns E_MACV error. Note, even if the user stack overflows while a task or task exception handling routine is executing, the error is not detected.

This feature is available only when the service calls is called from task context.

2.9.2 Domain, Memory Object, Access Permission Vector

The memory protection function is achieved by controlling the following.

- Who
- To where
- What access is permitted

The one that corresponds to "Who" is domain. Tasks and task exception handling routines belong to either of domain without fail. Domains are distinguished by domain ID with the value from 1 to 15. Domains is generated statically by cfg file.

The one that corresponds to "To where" is memory object, and the one that corresponds to "What access is permitted" is access permission vector.

Usually, memory objects are registered statically by cfg file. Memory objects can be registered dynamically by using ata_mem service call and unregistered by using det_mem service call. The start address of a memory object must be 16-bytes boundary, and the size must be multiple of 16.

Access permission vector represents whether tasks that belong to each domain can access (operand read, operand write, execute) to the memory object.

The access exception handler will be initiated if a task accesses to the memory object that has not been permitted, or a task accesses other than memory objects and user stack for itself.

On the other hand, in handlers (interrupt handlers, cyclic handlers, alarm handlers and access exception handler), there is no concept of belonging domain. Handlers can access all address space.

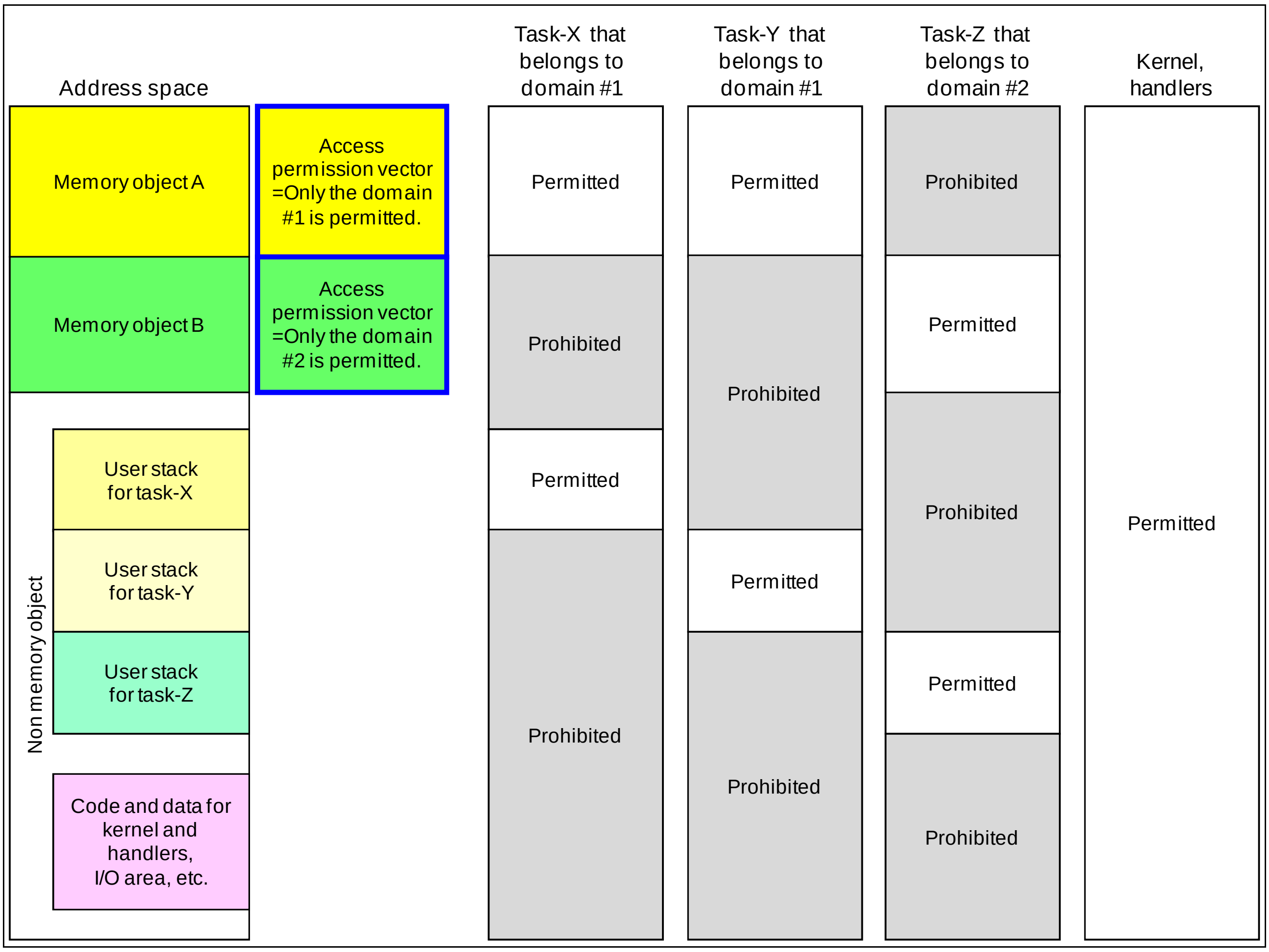

Figure 2.12 Summary of Memory Protection

Table 2.3 shows the operation concerning memory objects.

Table 2.3 The Operation Concerning Memory Objects

Operation	Static (cfg file)	Dynamic (service call)
Registration	memory_object[]	ata_mem
Unregistration	-	det_mem
Change access permission	-	sac_mem
Checks access permission (Other than memory object can be checked.)	-	vprb_mem
Refers to state	-	ref_mem

2.9.3 Restriction in the Number of Memory Objects

The number of memory objects to which either of access (operand read, operand write and execution) has been permitted by a certain domain is seven or less. Please design memory map nothing this requirement.

The E_MACV error is returned if the operation that doesn't fill this restriction is done.

2.9.4 Static Memory Object Registration Requirements

When a memory object is registered by memory_object[] statement in the cfg file, the memory object address can be specified by absolute address or section name.

The absolute address is used for registration of I/O area.

In the using of section name, the head and last section should be specified.

When specifying section name, it is necessary to arrange sections according to the assumption at linking. For example, specify "aligned_section" linker option to the head section of memory object (memory_object[].start_address), because the start address of memory objects must be 16-bytes boundary.

And the size of memory objects must be multiple of 16. This means that the termination address of memory object must be 16-bytes boundary + 15. However, the termination address of memory object doesn't necessarily terminate as required. When the termination address of memory object is not 16-bytes boundary + 15, the address from termination address +1 to the next 16-bytes boundary+15 will be managed as part of the memory object. Therefore, no memory is to be allocated to the address from termination address + 1 to the next 16-bytes boundary + 15.

❑Example

When "CU_DOM1" is specified for "memory_object[].end_address" and the termination address of CU_DOM1 section is 0xFFFF1003, do not allocate any section within the range from 0xFFFF1004 to 0xFFFF100F.

When "aligned_section" option is specified for the section the CU_DOM1's following section at linking, no sections are arranged in the range from 0xFFFF1004 to 0xFFFF100F

2.9.5 Trusted Domain

The RI600/PX does not support other protection mechanism except memory protection. When the memory protection is used wrongly, the following example shows how illegal access is not detected.

1. The task-A with malice does not have access permission for the memory object-M.
2. The coder wrote the source code for Task-A to create and start the task-B that belongs to the domain which has access permission for the memory object-M.
3. The illegal access is not detected if the task-B accesses the memory object-M.

To prevent such illegal access, the RI600/PX supports "trusted domain". The following Service calls that gives the change to the composition of software as follows can be called only from tasks that belongs to trusted domain. The E_MACV error is returned when a tasks that does not belong to trusted domain calls either of these service calls.

- cre_???, acre_???, del_???, def_tex, ata_mem, det_mem, sac_mem

2.9.6 Changes Access Permission

The access permission for a memory object can be changed dynamically by using sac_mem.

For instance, the following example showcases the requirement to change the access permission

In this example, it is required download application into memory that belongs to another domain and execute the application as a task .

1. Register the area for downloading as a memory object (ata_mem). At his time, specify to permit access the memory object from the domain-A that the task to download belongs to. Afterwards, download to the memory object area.
2. After downloading, set to be able to access the memory object from the domain-B (sac_mem). And create and start the downloaded code as the task that belongs to the domain-B.

2.9.7 Protection of User Stack

The user stack of each task can be accessed only by the task. The access exception handler will be initiated if the user stack overflows or a task accesses the user stack for another task.

When service call invoked from task uses user stack, the kernel inspects whether stack pointer points in the range of the user stack for invoking task. If not, the error E_MACV is returned.

2.9.8 Checks Access Permission

In the program called from two or more domains, there is a scene that the program wants to judge whether the program can access the memory. In this scene, vprb_mem service call is useful. The vprb_mem inspects whether the specified task can do the specified access to the specified memory area

2.9.9 Access Exception Handler

The access exception handler will be initiated when a task or task exception handling routine accesses the memory that has not been permitted. For such situation, the access exception handler can either remove the factor of illegal access and return to normal program execution or be used for debug purposes.

2.9.10 Processor Mode

The memory protection by MPU (Memory Protection Unit) is effective only at user mode.

In the RI600/PX system, task context executes in user mode, and non-task context executes in supervisor mode.

2.9.11 Enables MPU

The kernel enables MPU at initiation (vsta_kn, ivsta_knl). Do not disable MPU after the kernel has been initialized as disabling MPU will result in the system operation cannot be guaranteed.

2.9.12 Area That Should Be the Inside of Memory Object

(1) Area that is accessed by tasks

Tasks can access only memory objects to which the permission is appropriately set except it's own user stack. Therefore, it is necessary that program sections, constant sections, uninitialized data sections and initialized data sections accessed by tasks should be allocated to the inside of memory objects. Moreover, when the task accesses I/O area, the I/O area should be the inside of memory objects.

(2) Message area handled by mailbox

The message must be generated in the memory objects that both transmitting task and receiving task can access.

However, the kernel management table exists in the top of message area. The system operation cannot be guaranteed if the management table is destroyed. For this reason, data queue or message buffer is recommended for message communication.

(3) Fixed-sized and variable-sized memory pool area

The memory pool area should be the inside of memory object which can be accessed by tasks that use memory blocks.

However, the kernel generates management tables in the memory pool area. The system operation cannot be guaranteed if the management table is destroyed.

- Create a fixed-sized memory pool statically by cfg file
  The fixed-sized memory pool area is generated in the section indicated by "memorypool[].section". When "memorypool[].section" is omitted, the fixed-sized memory pool area is generated in the "BURI_HEAP" section.
- Create a fixed-sized memory pool dynamically by using cre_mpf or acre_mpf
  Application should acquire the fixed-sized memory pool area, and specify the start address for cre_mpf or acre_mpf.
- Create a variable-sized memory pool statically by cfg file
  The variable-sized memory pool area is generated in the section indicated by "variable_memorypool[].mpl_section". When "variable_memorypool[].mpl_section" is omitted, the variable-sized memory pool area is generated in the "BURI_HEAP" section.
- Create a variable-sized memory pool dynamically by using cre_mpl or acre_mpl
  Application should acquire the variable-sized memory pool area, and specify the start address for cre_mpl or acre_mpl.

2.9.13 Area That Should Be the Outside of Memory Object

(1) RI600/PX sections other than BURI_HEAP

The RI600/PX sections other than BURI_HEAP should be allocated to the outside of memory objects because these sections are accessed only by the kernel. Refer to 7.4.2, "List of RI600/PX Sections."

(2) User stack area for tasks

The user stack area for tasks should be outside of memory objects. The correct system operation cannot be guaranteed if the user stack area overwraps with either all user stacks and memory objects.

- Create a task statically by cfg file
  The user stack area is generated in the section indicated by "task[].stack_section". When "task[].stack_section" is omitted, the user stack area is generated in the "SURI_STACK" section.
- Create a task dynamically by using cre_tsk or acre_tsk
  Application should acquire the user stack area, and specify the start address for cre_tsk or acre_tsk.

(3) Data queue area

Usually, the data queue area should be generated to the area other than memory objects and user stacks. When the data queue area is generated in the memory object, a task with the writing permission to the memory object might rewrite data queue area by mistake.

- Create a data queue statically by cfg file
  The data queue area is generated in the BRI_RAM section of RI600/PX.
- Create a data queue dynamically by using cre_dtq or acre_dtq
  Application should acquire the data queue area, and specify the start address for cre_dtq or acre_dtq.

(4) Message buffer area

Usually, the message buffer area should be generated to the area other than memory objects and user stacks. When the message buffer area is generated in the memory object, a task with the writing permission to the memory object might rewrite message buffer area by mistake.

- Create a message buffer statically by cfg file
  The message buffer area is generated in the section indicated by "message_buffer[].mbf_section". When "message_buffer[].mbf_section". is omitted, the message buffer area is generated in the "BRI_RAM" section.
- Create a message buffer dynamically by using cre_mbf or acre_mbf
  Application should acquire the message buffer area, and specify the start address for cre_mbf or acre_mbf.

(5) Fixed-sized memory pool management area

Usually, the fixed-sized memory pool management area should be generated to the area other than memory objects and user stacks. When the fixed-sized memory pool management area is generated in the memory object, a task with the writing permission to the memory object might rewrite the fixed-sized memory pool management area by mistake.

- Create a fixed-sized memory pool statically by cfg file
  The fixed-sized memory pool management area is generated in the BRI_RAM section of RI600/PX.
- Create a fixed-sized memory pool dynamically by using cre_mpf or acre_mpf
  Application should acquire the fixed-sized memory pool management area, and specify the start address for cre_mpf or acre_mpf.

2.10 Kernel Idling

When there is no READY task, the kernel enters an endless loop and waits for interrupts.

2.11 Task in Supervisor Mode

Tasks cannot run in the supervisor mode.

Please implement processing to run in the supervisor mode as interrupt handler for INT instrution.

For example, the WAIT instruction must be executed in the supervisor mode because the WAIT is privileged instruction.

Note, do not use from INT #1 to INT #8 because these are reserved by RI600/PX.

3. Kernel Functions

This section mainly describes the functions and usage of kernel service calls

3.1 Module Structure

The kernel consists of the modules shown in Figure 3.1. Each of these modules is composed of functions that exercise individual module features.

The kernel is supplied in the form of a library, and only necessary features are linked at the time of system generation. More specifically, only the functions used are chosen from those which comprise these modules and linked by means of the Linkage Editor. However, the scheduler module, part of the task management module, and part of the time management module are linked at all times because they are essential feature functions.

The applications program is a program created by the user. It consists of tasks, interrupt handler, cyclic handler, alarm handler, and access exception handler.

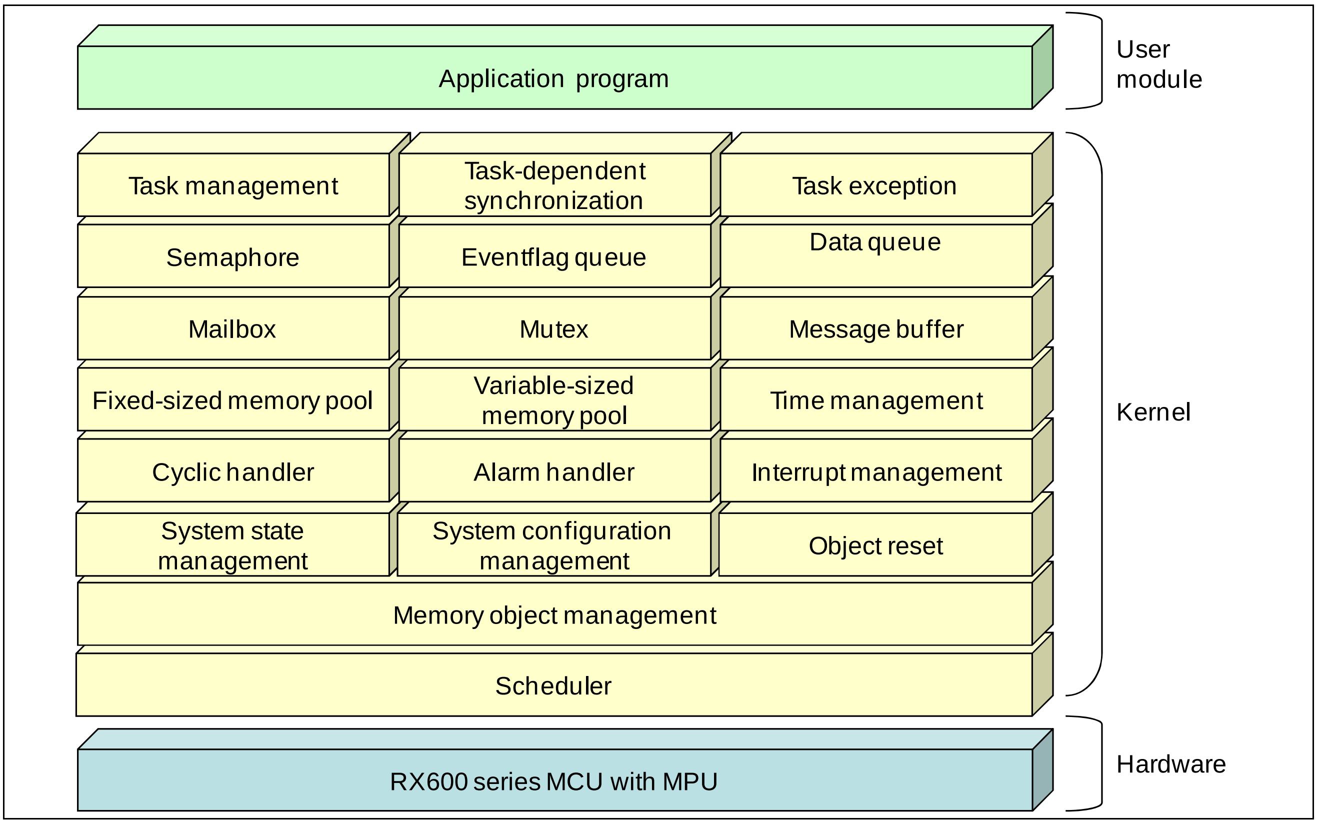

Figure 3.1 Kernel Structure

3.2 Module Overview

(1) Scheduler

Forms a task processing queue based on task priority and controls operation so that the high-priority task at the beginning in that queue (task with small priority value) is executed.

(2) Task management function

Exercises task operation such as creation, deletion, activation, termination, or changing task priority

(3) Task-dependent synchronization function

Accomplishes inter-task synchronization by changing the task status from a different task.

(4) Task exception handling function

Exercises task exception operation such as definition or generation.

(5) Synchronization and communication function

This is the function for synchronization and communication among the tasks by using the objects independent of task. The following five functional modules are offered.

- Semaphore
  The semaphore is the object to prevent resources between tasks from competing.
- Eventflag
  The eventflag is the object that controls the execution control of tasks according to the condition of AND/OR condition of the bit pattern.
- Data Queue
  The data queue is the object to communicate the 1-word (32 bits) data.
- Mailbox
  The mailbox is the object to communicate messages with arbitrary size. The message is not copied, the message address is transferred.
- Mutex
  The mutex is the object to prevent resources between tasks from competing. The mutex has the function to evade the priority inversion problem.
- Message Buffer
  The message buffer is the object to communicate messages with arbitrary size. The message is copied,

(6) Fixed-sized memory pool function

The fixed-sized memory pool is the object to allocate the fixed-sized memory block dynamically.

(7) Variable-sized memory pool function

The variable-sized memory pool is the object to allocate the memory block with arbitrary size dynamically.

(8) Interrupt management function

Makes a return from the interrupt handler.

(9) Time management function

Sets up the system timer used by the kernel and starts the user-created alarm handler and cyclic handler.

(10) System status management function

Changes or refers to the system state.

(11) System configuration management function

Gets kernel configuration information.

(12) Object reset function

Resets data queue, mailbox, message buffer, fixed-sized memory pool, and variable-sized memory pool.

This is the function outside µITRON 4.0 specification.

(13) Memory object management function

Protects memory object and user stack.

3.3 Task management Function

The task management functions are used to perform task operations such as creation, deletion, activating, ending tasks, and changing task priorities.

There are following method to create a task.

- Static creation
  Describe "task[]" statement in the cfg file.
- Dynamic creation
  Use cre_tsk or acre_tsk service call.

Following information are specified at creating a task.

- Task entry address
- Size of user stack
- Start address of user stack for dynamic creation, or section name of user stack for static creation
- Task initial priority
- Initial state after creation (TA_ACT attribute)
- Extended information
- Domain ID to which the task belongs

The kernel offers the following task management function service calls.

(1) Creates Task (cre_tsk, acre_tsk)

The cre_tsk creates a task with specified task ID. The acre_tsk creates a task and return the task ID. The created task enters to DORMANT state when TA_ACT attribute is not specified. When TA_ACT attribute is specified, the task is created and activated, and the task enters to READY state. The task activated by TA_ACT attribute receives the extended information.

For these service calls, start address of user stack and the size are specified. The application program must acquire the user stack with 16-bytes boundary address and 16-bytes boundary size.

The user stack area for tasks should be outside of memory objects. The correct system operation cannot be guaranteed if the user stack area overwraps with either all user stacks and memory objects.

The cre_tsk and acre_tsk can be issued from the task that belongs to the trusted domain.

(2) Deletes Task (del_tsk)

The del_tsk delets a task with specified task ID. The deleted task ID can be used by cre_tsk or acre_tsk.

The del_tsk can be issued from the task that belongs to the trusted domain.

(3) Activates Task (act_tsk, iact_tsk)

Activates the task with the specified ID. Unlike sta_tsk and ista_tsk, the activation requests by these service calls are queued, but a start code to be passed to the target task cannot be specified in these service calls. Extended information specified at the time of task creation is passed to the target task.

(4) Cancels Task Activation Requests (can_act, ican_act)

Cancels the activation requests that have been queued for the task with the specified ID.

(5) Starts Task (with start code) (sta_tsk, ista_tsk)

Activates the task with the specified ID. In either service call, unlike in act_tsk or iact_tsk, requests for service call startup of this type are not queued, but a start code to be passed to the target task can be specified.

(6) Terminates Invoking Task (ext_tsk)

Terminates the invoking task, placing the task in the DORMANT state. If activation requests for the task have been queued, task startup processing is performed again. In this case, the invoking task behaves as if it has been reset.

Behavior of the task in response to this service call is the same as the task returning from its entry function.

(7) Terminates and Deletes Invoking Task (exd_tsk)

Terminates the invoking task and delete the task. The deleted task ID can be used by cre_tsk or acre_tsk.

(8) Terminates Task (ter_tsk)

Terminates another task that is not in the DORMANT state and places the task in the DORMANT state. If activation requests for the task have been queued, task startup processing is performed again.

(9) Changes Task Priority (chg_pri, ichg_pri)

Changes the priority of the task with the specified ID. If the priority of a task is changed while the task is in the READY or RUNNING state, the ready queue is also updated (Figure 3.2). Moreover, if the target task is placed in the wait queue of an object with the TA_TPRI attribute, the wait queue is also updated (Figure 3.3).

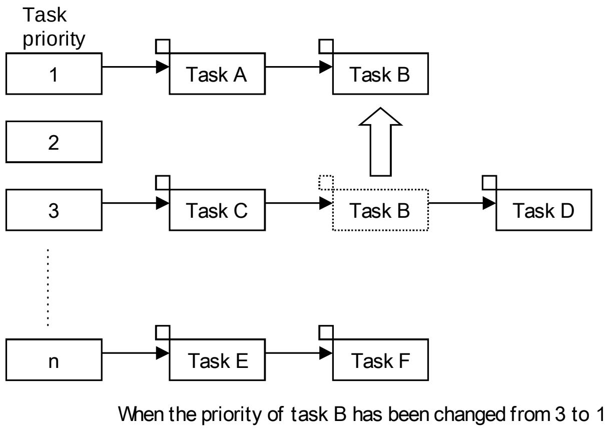

Figure 3.2 Ready Queue When Changing a Task Priority in the READY or RUNNING State

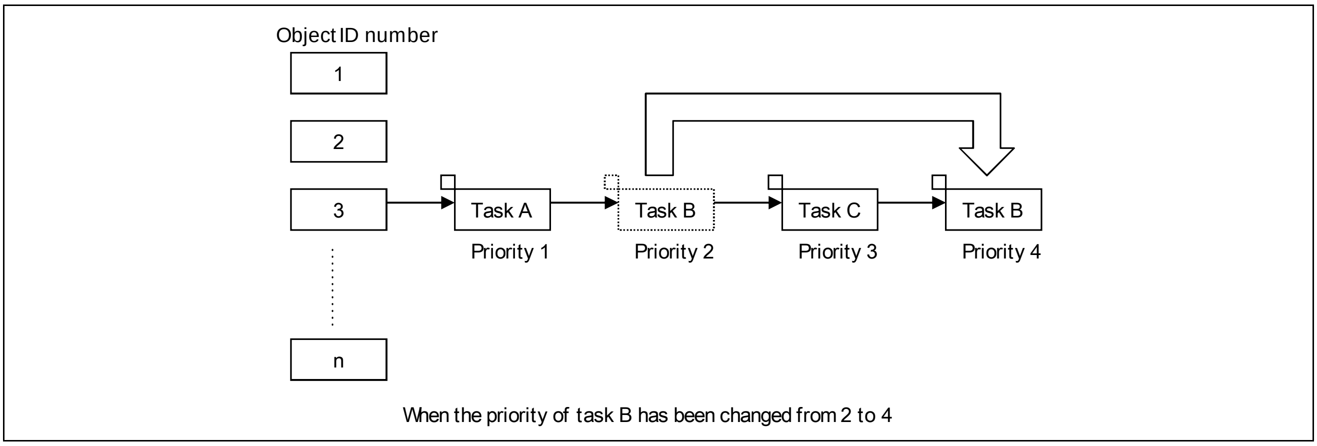

Figure 3.3 Re-Arranging the Wait Queue When Changing Priority of a task waiting for the object with TA_TPRI Attribute

However, it is generally recommended that these service calls not be used because changing the priority affects the behavior of the entire system.

A task has two priority levels: base priority and current priority. In general operation, these two priority levels are the same; they differ only while the task has a mutex locked. For details, refer to section 3.10, "Mutex."

(10) Refers to Task Priority (get_pri, iget_pri)

Gets the priority of the task with the specified ID.

(11) Refers to Task State (ref_tsk, iref_tsk)

Refers to state of the task with the specified ID.

(12) Refers to Task State (Simplified Version) (ref_tst, iref_tst)

Refers to state of the task with the specified ID. Either service call produces less overhead than ref_tsk or iref_tsk because it refers to less information.

3.4 Task-Dependent Synchronization Function

The task-dependent synchronization functions are used to achieve synchronization between tasks by placing tasks in the WAITING, SUSPENDED, or WAITING-SUSPENDED states, or to wake up tasks in the WAITING state.

The kernel offers the following task-dependent synchronization service calls.

(1) Puts Task to Sleep (slp_tsk, tslp_tsk) and Wakes up Task (wup_tsk, iwup_tsk)

The slp_tsk puts the current task to sleep. The task in sleep becomes to the WAITING state.

The tslp_tsk performs the same function as slp_tsk except that a timeout period before wakeup is specifiable. The wup_tsk or iwup_tsk wakes up a task from sleep state. The waked-up task is released from the WAITING state. While the target task is not in sleep state, the issued wakeup requests are queued. If the task for which wakeup requests have been queued calls slp_tsk or tslp_tsk, the wakeup request count is decremented by one and the task does not enter the WAITING state (Figure 3.4).

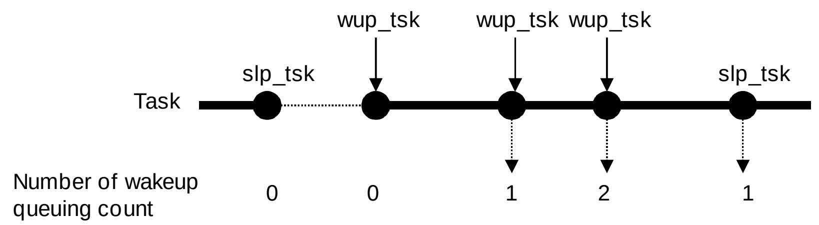

Figure 3.4 Queuing Wakeup Request

(2) Cancels Task Wakeup Request (can_wup, ican_wup)

Cancels the wakeup requests queued for a task with the specified ID (Figure 3.5).

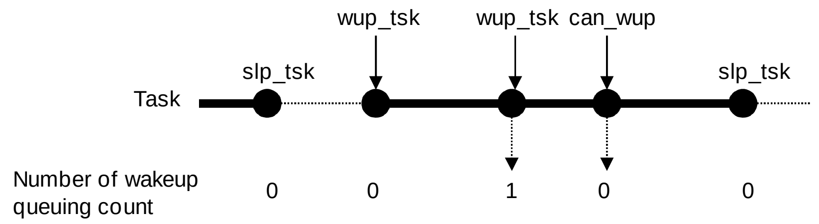

Figure 3.5 Canceling Wakeup Requests

(3) Suspends Task (sus_tsk, isus_tsk) and Resumes Task(rsm_tsk, irsm_tsk, frsm_tsk, ifrsm_tsk)

Issuing sus_tsk or isus_tsk forcibly suspends the task with the specified ID. If the target task is in the READY state, the task is placed in the SUSPENDED state. If the target task is in the WAITING state, the task is placed in the WAITING-SUSPENDED state.

Only one suspension request can be nested, if sus_tsk or isus_tsk is issued to a task in the SUSPENDED state, the error E_QOVR is returned.

The rsm_tsk or irsm_tsk decrements the suspension counts for a task with the specified ID. When the number reaches 0, the task is taken out of the SUSPENDED state (Figure 3.6).

The frsm_tsk or ifrsm_tsk clears the suspension count for a task with specified ID, and the task is taken out of the SUSPENDED state.

Because the maximum suspension count is 1, the behavior of the frsm_tsk and ifrsm_tsk is the same as the rsm_tsk and irsm_tsk.

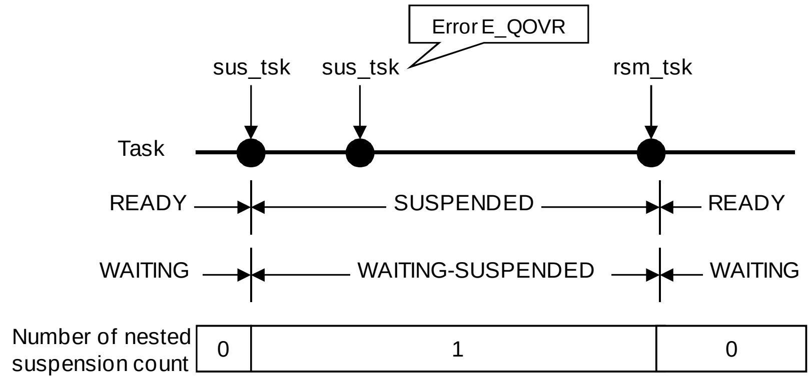

Figure 3.6 Suspending and Resuming Tasks

(4) Forcibly Releases Task from WAITING State (rel_wai, irel_wai)

The rel_wai or irel_wai forcibly releases the task with the specified ID from the WAITING state. Note that neither service call can release a task from the SUSPENDED state.

(5) Delays Task (dly_tsk)

The dly_tsk delays execution of the current task during the specified time. The current task enters to the WAITING state.

3.5 Task Exception Handling Function

When task exception is requested to a task, the task exception handling routine is initiated.

The task exception handling routine can be defined for each task.

There are following method to define a task exception handling routine.

- Static definition
  Describe "task[].texrtn" statement in the cfg file.
- Dynamic definition
  Use def_tex service call.

Following information are specified at defining a task exception handling routine.

- Task exception handling routine entry address

When all of following conditions are satisfied, the execution of corresponded task is discontinued, and the task exception handling routine for corresponded task is initiated.

1. Corresponded task is in task exception enabled state.
2. Pended exception code is not 0.
3. Non-task context is not executed.
4. Corresponded task is in RUNNING state.

The exception code is represented by bit pattern. When task exception is requested (ras_tex, iras_tex), the pending exception code for corresponded task is renewed to the logical add with specified exception code.

When a task exception handling routine is initiated, the task exception is changed to disabled state, and all its of the exception code are cleared. The task exception handling routine receives exception code before clear and the extended information for the task.

When a task exception handling routine is finished, the task exception is changed to enabled state, and the task execution continues from discontinued point.

At the task initiation, the task exception is in disabled state. To be enable the task exception, the task should calls ena_tex.

- Operation that be disable the task exception
  (1) Start of task
  (2) dis_tex
  (3) Start of the task exception handling routine
  (4) Release the definition of the task exception handling routine by using def_tex
- Operation that be enable the task exception
  (1) ena_tex
  (2) Finish of the task exception handling routine

" Corresponded task is in RUNNING state" in the above-mentioned four start conditions means scheduling the task. Table 3.1shows operations with the possibility of meeting the start condition excluding this.

Table 3.1 Start Condition of Task Exception Handling Routine

Operations with the possibility of meeting the start condition	Start condition		
	1. Corresponded task is in task exception handling enabled state.	2. Pending exception code is not 0.	3. Non-task context is not executed.
ras_tex, iras_tex		√	
ena_tex	√		
Finish of task exception handling routine	√		
Finish of interrupt handler			√

The kernel offers the following task exception function service calls.

(1) Defines Task Exception Handling Routine and release the definition (def_tex)

The def_tex defines the task exception handling routine for the task with specified task ID. Moreover, the def_tex can release the definition.

The def_tex can be issued from the task that belongs to the trusted domain.

(2) Raises Task Exception Handling (ras_tex, iras_tex)

The ras_tex and iras_tex request task exception with specified exception code for the task with specified task ID.

(3) Disables Task Exception (dis_tex)

The dis_tex shifts the current task to the task exception disabled state

(4) Enables Task Exception (ena_tex)

The ena_tex shifts the current task to the task exception enabled state

(5) Refers to Task Exception Disabled State (sns_tex)

The sns_tex refers to whether the current task is in the task exception disabled state.

(6) Refers to Task Exception State (ref_tex, iref_tex)

The ref_tex and iref_tex refers to the task exception state for the task with specified task ID.

3.6 Semaphore

3.6.1 Functions

The semaphore is object used to prevent conflicts over resources such as devices or variables shared by multiple tasks. For example, if task switching occurs while task A is updating a shared variable and task B refers to this variable when updating of its value is not complete, task B may incorrectly read the shared variable. Such conflicts can be prevented by using semaphores.

A semaphore provides exclusive control and a synchronization function by expressing the existence of a resource or the number of resources as a counter.

Applications must be programmed so that semaphores are associated with resources to be exclusively controlled.

Note the following rules on exclusive control using a semaphore.

(1) A task should acquire the semaphore before using the associated resource
(2) A task should release the semaphore after its usage of the resource is finished

There are following method to create a semaphore.

- Static creation
  Describe "semaphore[]" statement in the cfg file.
- Dynamic creation
  Use cre_sem or acre_sem service call.

Following information are specified at creating a semaphore.

- How tasks are queued waiting for the semaphore
  TA_TFIFO (queued in FIFO order) or TA_TPRI (queued in order of task priority)
- Initial semaphore resource count
- Maximum semaphore resource count

Figure 3.7 shows an example of semaphore usage.

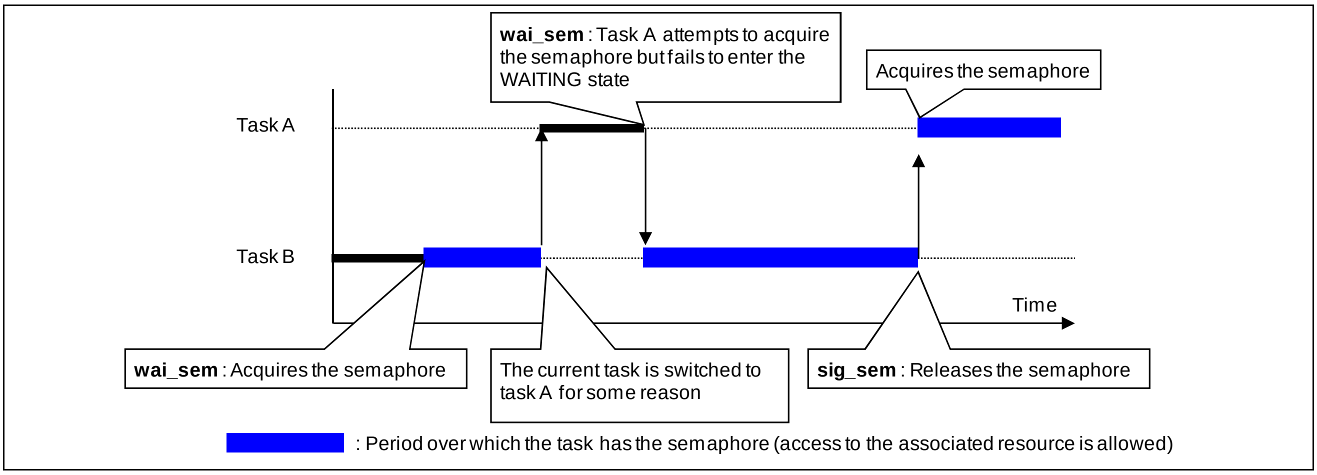

Figure 3.7 Example of Semaphore Usage

The kernel offers the following semaphore function service calls.

(1) Creates Semaphore (cre_sem, acre_sem)

The cre_sem creates a semaphore with specified semaphore ID. The acre_sem creates a semaphore and return the semaphore ID.

The cre_sem and acre_sem can be issued from the task that belongs to the trusted domain.

(2) Deletes Semaphore (del_sem)

The del_sem deletes a semaphore with specified semaphore ID. The deleted semaphore ID can be used by cre_sem or acre_sem.

The del_sem can be issued from the task that belongs to the trusted domain.

(3) Acquires Semaphore Resource (wai_sem, twai_sem)

Acquires a semaphore. If the semaphore's resource count has a positive value, the resource count is decremented by one. If the semaphore cannot be acquired (semaphore's resource count = 0), the task enters the WAITING state.

(4) Acquires Semaphore Resource (Polling) (pol_sem, ipol_sem)

Acquires a semaphore. The only difference between these service calls and wai_sem or twai_sem is that an error is immediately returned and the task does not enter the WAITING state when the semaphore count is 0.

(5) Releases Semaphore Resource (sig_sem, isig_sem)

Releases a semaphore. When a task is waiting to acquire a semaphore, either service call makes the task leave the WAITING state. If not, the resource count is incremented by one.

(6) Refers to Semaphore Status (ref_sem, iref_sem)

Refers to the state of a semaphore, including its resource count and the IDs of waiting tasks.

3.6.2 Priority Inversion Problem

When a semaphore is used for exclusive control of a resource, a problem called priority inversion may arise. This refers to the situation where a task that is not using a resource delays the execution of a task requesting the resource.

Figure 3.8 illustrates this problem. In this figure, tasks A and C are using the same resource, which task B does not use. Task A attempts to acquire a semaphore so that it can use the resource but enters the WAITING state because task C is already using the resource. Task B has a priority higher than task C and lower than task A. Thus, if task B is executed before task C has released the semaphore, release of the semaphore is delayed by the execution of task B. This also delays acquisition of the semaphore by task A. From the viewpoint of task A, a lower-priority task that is not even competing for the resource gets priority over task A.

To avoid this problem, use a mutex instead of a semaphore.

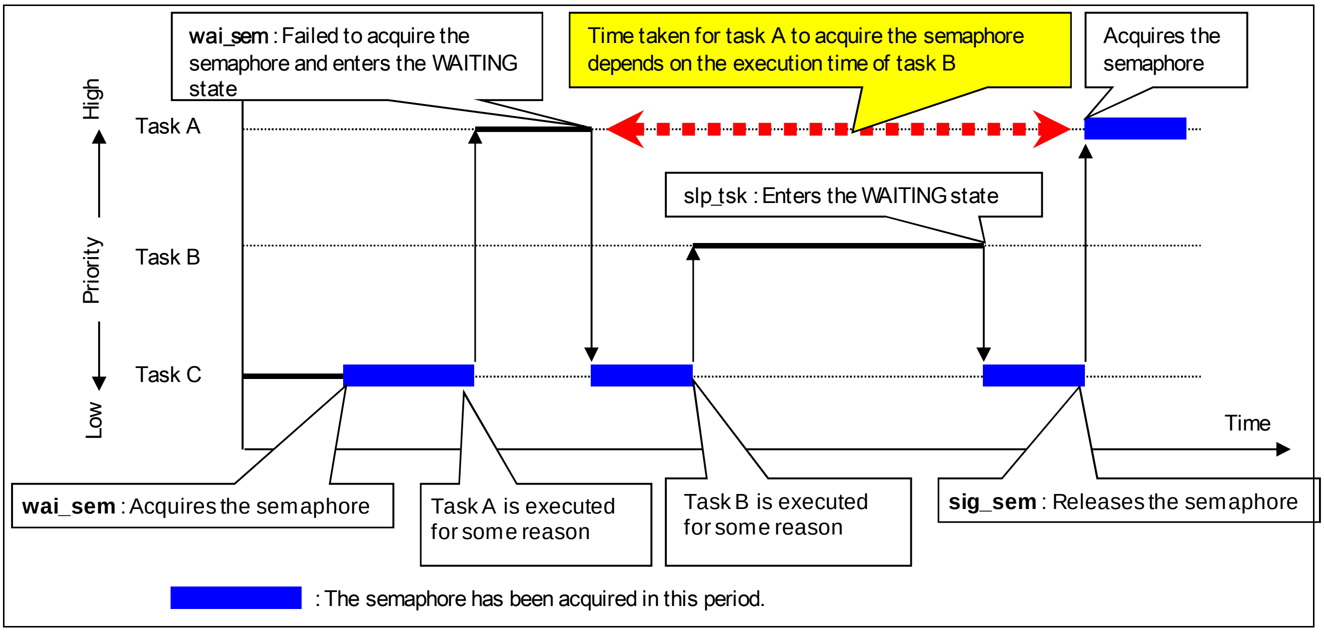

Figure 3.8 Priority Inversion Problem

3.7 Eventflag

The eventflag is group of bits that correspond to events. One event corresponds to one bit. A task can be made to wait for one (OR condition) or all (AND condition) of the specified bits to be set. Whether more than one task is allowed to wait for a specific bit of an eventflag to be set can be selected as an attribute when the eventflag is created. Either of the following attributes is selectable.

- TA_WMUL : Multiple tasks are permitted to wait
- TA_WSGL : Multiple tasks not permitted to wait

A TA_CLR attribute is also specifiable; in this case, the bit pattern of the eventflag is cleared to 0 whenever the wait condition of a task is satisfied.

One feature of the eventflag mechanism is that multiple tasks can be released from the WAITING state at the same time. To allow this, specify the TA_WMUL attribute. Do not specify the TA_CLR attribute in this case.

Figure 3.9 shows an example of task execution control by an eventflag. In this figure, six tasks, task A to task F, have been placed in a wait queue. After the flag pattern has been set to 0x0F by the service call set_flg, the pattern satisfies the wait conditions for three of the tasks (task A, task C, and task E). These tasks are sequentially removed from the head of the queue.

If this eventflag has the TA_CLR attribute, when task A is released from the WAITING state, the bit pattern of the eventflag will be set to 0, and task C and task E will not be removed from the queue.

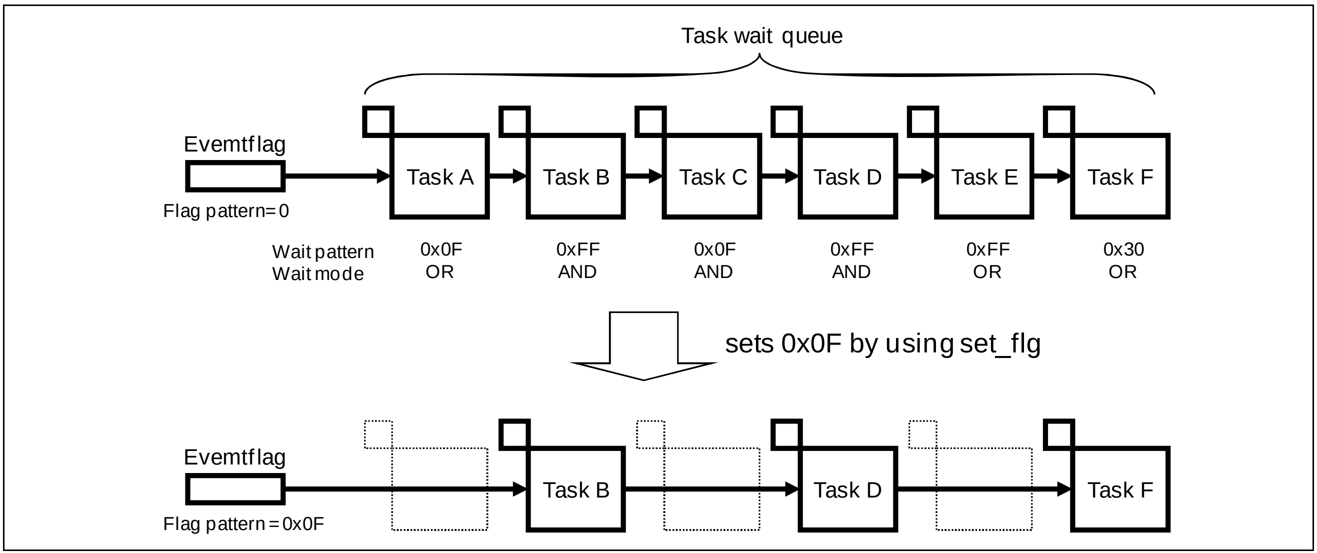

Figure 3.9 Example of Eventflag Usage

There are following method to create a semaphore.

- Static creation
  Describe "flag[]" statement in the cfg file.
- Dynamic creation
  Use cre_flg or acre_flg service call.

Following information are specified at creating a eventflag.

- How tasks are queued waiting for the semaphore
  TA_TFIFO (queued in FIFO order) or TA_TPRI (queued in order of task priority)
- Initial pattern of eventflag
- Multiple wait permission attribute
  TA_WMUL (multiple tasks are permitted to wait) or TA_WSGL (multiple tasks not permitted to wait)
- Clear attribute (TA_CLR)

The kernel offers the following eventflag function service calls.

(1) Creates Eventflag (cre_flg, acre_flg)

The cre_flg creates a eventflag with specified eventflag ID. The acre_flg creates a eventflag and return the eventflag ID.

The cre_flg and acre_flg can be issued from the task that belongs to the trusted domain.

(2) Deletes Eventflag (del_flg)

The del_flg deletes a eventflag with specified eventflag ID. The deleted eventflag ID can be used by cre_flg or acre_flg.

The del_flg can be issued from the task that belongs to the trusted domain.

(3) Waits for Eventflag (wai_flg, twai_flg)

Makes a task wait until specific bits in the eventflag have been set. Select either of the following wait conditions.

- AND condition : Waits until all of the specified bits have been set
- OR condition : Waits until any of the specified bit have been set

When a task is released from the WAITING state, the value of the eventflag at satisfaction of the wait condition is returned to the task that issued this service call. If the TA_CLR attribute has been specified for the eventflag, the eventflag is also cleared to 0. In this case, the value of the eventflag immediately before it was cleared is returned to the task that issued this service call.

(4) Acquires Eventflag (Polling) (pol_flg, ipol_flg)

Checks if specified bits in an eventflag have been set. The only difference between these service calls and wai_flg or twai_flg is that an error code is immediately returned and the task does not enter the WAITING state when the condition is not satisfied.

(5) Sets Eventflag (set_flg, iset_flg)

Sets an eventflag to a specified bit pattern. This may release tasks with wait conditions that match the pattern.

(6) Clears Eventflag (clr_flg, iclr_flg)

Clears specified bits of an eventflag.

(7) Refers to Eventflag Status (ref_flg, iref_flg)

Refers to the state of an eventflag, including its bit pattern and the IDs of waiting tasks.

3.8 Data Queue

The data queue is object used to achieve the communication of single word (32-bit units) of data.

Figure 3.10 shows the structure of a data queue.

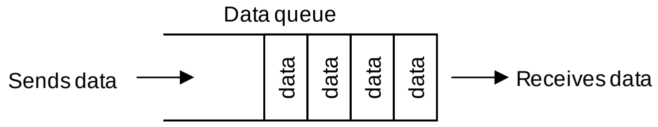

Figure 3.10 Data Queue

Data are sent to a data queue for storage. When data are received from a data queue, the oldest data are taken out first (on an FIFO basis). The maximum number of data items that can be queued in a data queue is specifiable when the data queue is created. A data queue with no storage can be used.

There are following method to create a data queue.

- Static creation
  Describe "dataqueue[]" statement in the cfg file.
- Dynamic creation
  Use cre_dtq or acre_dtq service call.

Following information are specified at creating a data queue.

- How tasks are queued waiting to send
  TA_TFIFO (queued in FIFO order) or TA_TPRI (queued in order of task priority)
- Number of data items that can be stored
- Start address of the data queue area for dynamic creation

The kernel offers the following data queue function service calls.

(1) Creates Data Queue (cre_dtq, acre_dtq)

The cre_dtq creates a data queue with specified data queue ID. The acre_dtq creates a data queue and return the data queue ID.

Usually, the data queue area should be generated to the area other than memory objects and user stacks. When the data queue area is generated in the memory object, a task with the writing permission to the memory object might rewrite data queue area by mistake.

The cre_dtq and acre_dtq can be issued from the task that belongs to the trusted domain.

(2) Deletes Eventflag (del_dtq)

The del_dtq deletes a data queue with specified data queue ID. The deleted data queue ID can be used by cre_dtq or acre_dtq.

The del_dtq can be issued from the task that belongs to the trusted domain.

(3) Sends to Data Queue (snd_dtq, tsnd_dtq)

Sends a data to an data queue. When the data queue is full of data, the calling task enters the WAITING state.

(4) Sends to Data Queue (Polling) (psnd_dtq, ipsnd_dtq)

Sends a data to a data queue. The only difference between these service calls and snd_dtq or tsnd_dtq is that an error code is immediately returned and the calling task does not enter the WAITING state when the data queue is full.

(5) Forcibly Sends to Data Queue (fsnd_dtq, ifsnd_dtq)

Sends a data to a data queue. When the data queue is full of data, the oldest data are deleted and the new data are sent.

(6) Receives from Data Queue (rcv_dtq, trcv_dtq)

Receives a data from a data queue. When the data queue has no data, the calling task enters the WAITING state. If the data queue was full of data and a task was waiting to send data, this call releases the first task in the wait queue for sending data from the WAITING state.

(7) Receives from Data Queue (Polling) (prcv_dtq, iprcv_dtq)

Receives a data from a data queue. When the data queue has no data, an error code is returned. If the data queue was full of data and a task was waiting to send data, this call releases the first task in the wait queue for sending data from the WAITING state.

(8) Refers to Data Queue Status (ref_dtq, iref_dtq)

Refers to the state of a data queue, including the number of data stored in the queue, the ID of task waiting to send, and the ID of task to receive data.

3.9 Mailbox

3.9.1 Functions

The mailbox is object used to send or receive messages, which are data with arbitrary size.

Figure 3.11 shows the structure of a mailbox.

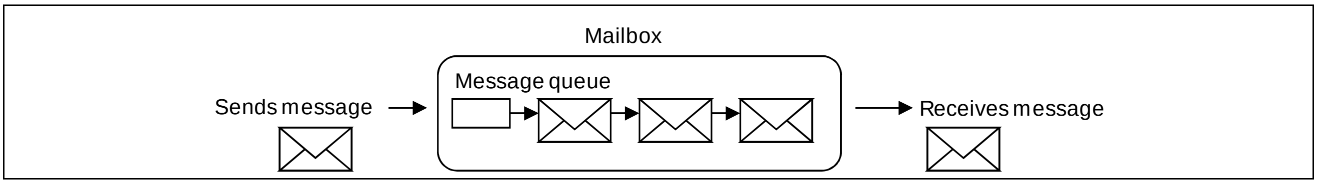

Figure 3.11 Mailbox

High-speed data communications are achieved regardless of the message size because only the addresses where the messages start are sent and received. Applications should create messages in memory areas that are accessible by both the sending and receiving tasks (i.e., messages should not be created in the local variable areas). A sending task should not access the message area after it has sent the message.

Messages transmitted to mailbox are managed by message queue. The messages are received in order of message queue.

Either the following can be selected as the order of mailbox.

- TA_MPRI attriburte : Queued in order of message's priority
- TA_MFIFO attribute : Queued in FIFO order

There are following method to create a mailbox.

- Static creation
  Describe "mailbox[]" statement in the cfg file.
- Dynamic creation
  Use cre_mbx or acre_mbx service call.

Following information are specified at creating a mailbox.

- How tasks are queued waiting to receive
  TA_TFIFO (queued in FIFO order) or TA_TPRI (queued in order of task priority)
- How messages are queued
  TA_MFIFO (queued in FIFO order) or TA_MPRI (queued in order of message's priority)
- Maximum message priority

The kernel offers the following mailbox function service calls.

(1) Creates Mailbox (cre_mbx, acre_mbx)

The cre_mbx creates a mailbox with specified mailbox ID. The acre_mbx creates a mailbox and return the mailbox ID.

The cre_mbx and acre_mbx can be issued from the task that belongs to the trusted domain.

(2) Deletes Mailbox (del_mbx)

The del_mbx deletes a mailbox with specified mailbox ID. The deleted mailbox ID can be used by cre_mbx or acre_mbx.

The del_mbx can be issued from the task that belongs to the trusted domain.

(3) Sends to Mailbox (snd_mbx, isnd_mbx)

Sends a message to a mailbox.

(4) Receives from Mailbox (rcv_mbx, trcv_mbx)

Receives a message from a mailbox. When the mailbox has no message, the task is in the WAITING state until a message is sent to the mailbox.

(5) Receives from Mailbox (Polling) (prcv_mbx, iprcv_mbx)

Receives a message from a mailbox. The only difference between these service calls and rcv_mbx or trcv_mbx is that an error code is immediately returned and the task does not enter the WAITING state when the mailbox has no message.

(6) Refers to Mailbox Status (ref_mbx, iref_mbx)

Refers to the address of the first message queued in the mailbox and the IDs of waiting tasks.

3.9.2 Notes

(1) Mailbox is not recommended to use

There is a kernel management table at the top of the message. There is danger destroyed by application because this management table is not protected. It is recommended for message communication to use data queue or message buffer instead of mailbox.

(2) Message location

In using mailbox, pointer to the message is transmitted, the message contents are not transmitted. Therefore, the message must be generated in the area which can be accessed from both sending task and receiving task. This means that the message must be generated in the memory object which can be accessed from both sending task and receiving task.

A bad example is shown as follows.

- bad example : generate a message as auto variable

  The auto variables are assigned to the stack frame for the function. When the function which sent a message finishes, the stack frame is released. After that, the old stack frame area is used (over-write) for another purpose. In a word, the message contents has been over-written at receiving the message.

  In addition, in the viewpoint of memory protection, task cannot access user stack for another task. So, receiving task cannot access received message,

3.10 Mutex

3.10.1 Functions

The mutex is object used to achieve exclusive control. It differs from a semaphore on the following points.

(1) A priority ceiling protocol is applied to avoid priority inversion problems.
(2) A mutex can only be used for exclusive control of a single resource.

Application should lock a mutex before using the related resource, and unlock the mutex after having finished using the resource. The priority of the task raised to the ceiling priority of the mutex. therefore the priority inversion problem is evaded.

Original behavior of the priority ceiling protocol is to make the current priority of the task to the highest ceiling priority of mutexes which are locked by the task. This behavior is achieved by controlling the current priority of the task as follows.

- When a task locks a mutex, changes the current priority of the task to the highest ceiling priority of mutexes which are locked by the task.
- When a task unlocks a mutex, changes the current priority of the task to the highest ceiling priority of mutexes which continues to be locked by the task. When there is no mutex locked by the task after unlock, returns the current priority of the task to the base priority.

However, the kernel adopts simplified priority ceiling protocol because of reducing overhead. Therefore, the underlined part is not processed.

Figure 3.12 shows an example of mutex usage.

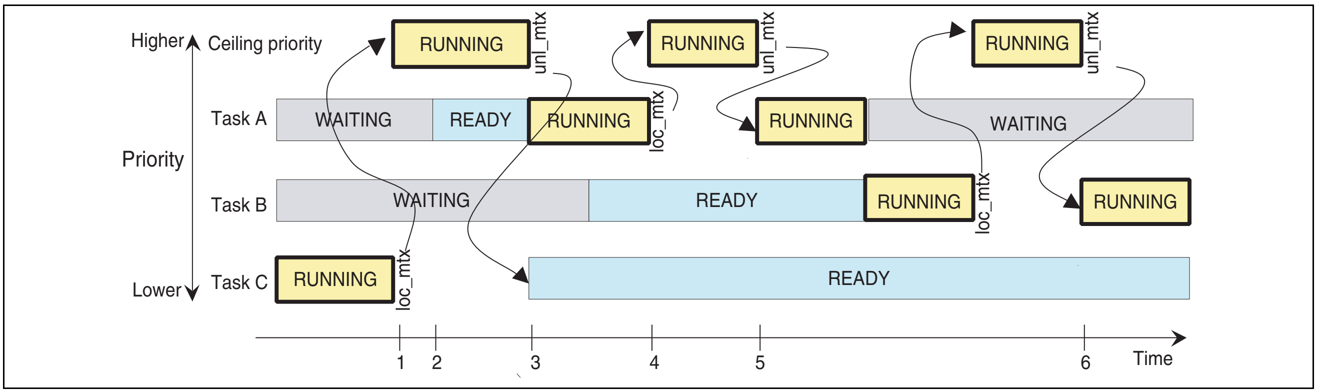

Figure 3.12 Example of Mutex Usage

Description:

1 Task C locks a mutex by issuing loc_mtx. The priority of task C is raised to the ceiling priority specified for the mutex.
2 Task A enters the READY state while task C is being executed at the ceiling priority. The priority of task A is higher than that initially specified for task C. However, task C now locks the mutex and is thus executed at the ceiling priority. Since this is higher than that of task A, task A cannot enter the RUNNING state. In other words, while task C has the mutex locked, execution of task C continues even if task A, with its higher initial priority, becomes ready.
3 Task C unlocks the mutex by issuing unl_mtx. The priority of task C returns to the initial level and higher-priority task A enters the RUNNING state.
4 Task A issues loc_mtx to raise its priority to the ceiling priority.
5 Task A issues unl_mtx to return its priority to the initial level.
6 Task B issues loc_mtx to raise its priority to the ceiling priority.
7 Task B issues unl_mtx to return its priority to the initial level.

There are following method to create a mutex.

- Static creation
  Describe "mutex[]" statement in the cfg file.
- Dynamic creation
  Use cre_mtx or acre_mtx service call.

Following information are specified at creating a mutex.

- Ceiling priority

The kernel offers the following mutex function service calls.

(1) Creates Mutex (cre_mtx, acre_mtx)

The cre_mtx creates a mutex with specified mutex ID. The acre_mtx creates a mutex and return the mutex ID.

The cre_mtx and acre_mtx can be issued from the task that belongs to the trusted domain.

(2) Deletes Mutex (del_mtx)

The del_mtx deletes a mutex with specified mutex ID. The deleted mutex ID can be used by cre_mtx or acre_mtx.

The del_mtx can be issued from the task that belongs to the trusted domain.

(3) Locks Mutex (loc_mtx, tloc_mtx)

Locks a mutex and raises the current priority of the locking task to the ceiling priority of the mutex. When another task has already locked the mutex, the task that issued loc_mtx or tloc_mtx enters the WAITING state until the mutex is unlocked.

(4) Locks Mutex (Polling) (ploc_mtx)

Locks a mutex and raises the current priority of the locking task to the ceiling priority of the mutex. The only difference between this service call and loc_mtx or tloc_mtx is that an error is immediately returned and the task that issued ploc_mtx does not enter the WAITING state when another task has already locked the mutex.

(5) Unlocks Mutex (unl_mtx)

Unlocks a mutex. When invoking task have not locked another mutexes, the current priority of the task returns the base priority. When a task is waiting to lock the mutex, this service call makes the task leave the WAITING state.

(6) Refers to Mutex Status (ref_mtx)

Refers to the state of a mutex, including the ID of a task that has locked the mutex and of waiting tasks.

3.10.2 Base Priority and Current Priority

A task has two priority levels: base priority and current priority. Tasks are scheduled according to current priority.

While a task does not have a mutex locked, its current priority is always the same as its base priority.

When a task locks a mutex, only its current priority is raised to the ceiling priority of the mutex.

When priority-changing service call chg_pri or ichg_pri is issued, both the base priority and current priority are changed if the specified task does not have a mutex locked. When the specified task locks a mutex, only the base priority is changed. When the specified task has a mutex locked or is waiting to lock a mutex, an E_ILUSE error is returned if a priority higher than the ceiling priority of the mutex is specified.

The current priority can be checked through service call get_pri or iget_pri.

3.11 Message Buffer

Like mailbox, the message buffer is object for the sending and receiving of messages, which are data with arbitrary size. The only difference is that the actual contents of the messages are copied and passed. For this reason, the message area becomes available immediately after a message has been sent, regardless of whether or not the receiving task has received the message.

Figure 3.13 shows the structure of a message buffer.

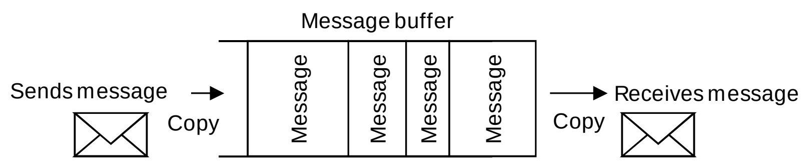

Figure 3.13 Message Buffer

Messages sent to a message buffer are stored in the buffer. When a message is received from a message buffer, the oldest message is taken out first (i.e. operation is FIFO).

There are following method to create a message buffer.

- Static creation
  Describe "message_buffer[]" statement in the cfg file.
- Dynamic creation
  Use cre_mbf or acre_mbf service call.

Following information are specified at creating a message buffer.

- Buffer size
- Start address of buffer area for dynamic creation, or section name of the buffer area for static creation
- Maximum size of a message that can be transmitted

The kernel offers the following message buffer function service calls.

(1) Creates Message Buffer (cre_mbf, acre_mbf)

The cre_mbf creates a message buffer with specified message buffer ID. The acre_mbf creates a message buffer and return the message buffer ID.

Usually, the message buffer area should be generated to the area other than memory objects and user stacks. When the message buffer area is generated in the memory object, a task with the writing permission to the memory object might rewrite message buffer area by mistake.

The cre_mbf and acre_mbf can be issued from the task that belongs to the trusted domain.

(2) Deletes Message Buffer (del_mbf)

The del_mbf deletes a message buffer with specified message buffer ID. The deleted message buffer ID can be used by cre_mbf or acre_mbf.

The del_mbf can be issued from the task that belongs to the trusted domain.

(3) Sends to Message Buffer (snd_mbf, tsnd_mbf)

Sends a message to a message buffer.

For a message to be sent to a massage buffer, the message buffer must have at least the following amount of free space:

```
(Size of the message in bytes rounded up to a multiple of 4) + VTSZ_MBFTBL
```

When the message buffer has less free space than is required, the task is in the WAITING state until enough space becomes available. This wait queue is managed in FIFO order.

(4) Sends to Message Buffer (Polling) (psnd_mbf, ipsnd_mbf)

Sends a message to a message buffer. The only difference between these service calls and snd_mbf or tsnd_mbf is that an error code is immediately returned and the task does not enter the WAITING state when the message buffer does not have enough free space.

(5) Receives from Message Buffer (rcv_mbf, trcv_mbf)

Receives a message from a message buffer. When the message buffer has no messages, the task enters the WAITING state until a message is sent to the message buffer. This wait queue is managed in FIFO order.

When a message is received from the message buffer, free space in the message buffer increases by the following amount:

```
(Size of the message in bytes rounded up to a multiple of 4) + VTSZ_MBFTBL
```

When the amount of free space in the message buffer becomes larger than the size of the message that a task is waiting to send, the message is sent to the message buffer and the task leaves the WAITING state.

(6) Receives from Message Buffer (Polling) (prcv_mbf)

Receives a message from a message buffer. The only difference between this service call and rcv_mbf or trcv_mbf is that an error code is immediately returned and the task does not enter the WAITING state when the message buffer has no messages.

(7) Refers to Message Buffer Status (ref_mbf, iref_mbf)

Refers to the state of a message buffer, including the number of messages it contains, the amount of free space, and the IDs of tasks waiting to send or receive messages.

3.12 Fixed-sized Memory Pool

3.12.1 Functions

The fixed-sized memory pool is object used to dynamically allocate and release memory blocks of fixed size. While fixed-sized memory pools cannot be used to acquire memory blocks of arbitrary size, their advantage over variable-sized memory pools is that acquiring and releasing blocks produces less overhead.

Figure 3.14 is a schematic view of a fixed-sized memory pool.

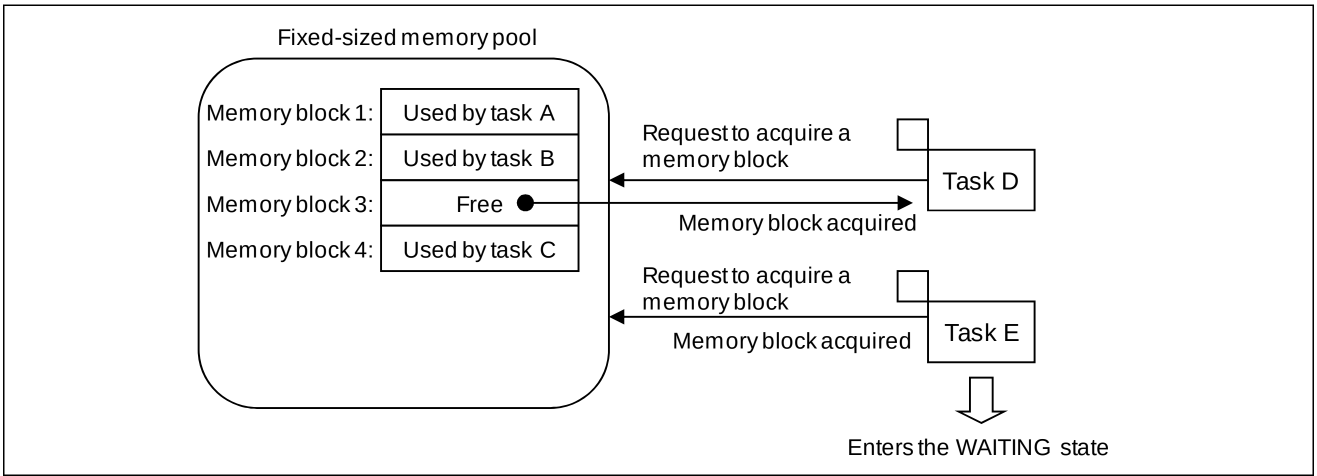

Figure 3.14 Fixed-sized Memory Pool

There are following method to create a fixed-sized memory pool.

- Static creation
  Describe "memorypool[]" statement in the cfg file.
- Dynamic creation
  Use cre_mpf or acre_mpf service call.

Following information are specified at creating a fixed-sized memory pool.

- How tasks are queued waiting to acquire memory block
  TA_TFIFO (queued in FIFO order) or TA_TPRI (queued in order of task priority)
- Size of memory block
- Number of memory blocks
- Start address of memory pool area for dynamic creation, or section name of the memory pool area for static creation
- Start address of the fixed-sized memory pool management area for dynamic creation

The kernel offers the following fixed-sized memory pool function service calls.

(1) Creates Fixed-sized Memory Pool (cre_mpf, acre_mpf)

The cre_mpf creates a fixed-sized memory pool with specified fixed-sized memory pool ID. The acre_mpf creates a fixed-sized memory pool and return the fixed-sized memory pool ID.

The fixed-sized memory pool area should be the inside of memory object which can be accessed by tasks that use memory blocks. Two or more fixed-sized/variable-sized memory pools may be put in the same memory object. Note, different access permission at every memory block cannot be set.

Usually, the fixed-sized memory pool management area should be generated to the area other than memory objects and user stacks. When the fixed-sized memory pool management area is generated in the memory object, a task with the writing permission to the memory object might rewrite the fixed-sized memory pool management area by mistake.

The cre_mpf and acre_mpf can be issued from the task that belongs to the trusted domain.

(2) Deletes Fixed-sized Memory Pool (del_mpf)

The del_mpf deletes a fixed-sized memory pool with specified fixed-sized memory pool ID. The deleted fixed-sized memory pool ID can be used by cre_mpf or acre_mpf.

The del_mpf can be issued from the task that belongs to the trusted domain.

(3) Gets Fixed-sized Memory Block (get_mpf, tget_mpf)

Acquires a fixed-sized memory block. When no memory block is available in the memory pool, the task enters the WAITING state until a memory block is released.

(4) Gets Fixed-sized Memory Block (Polling) (pget_mpf, ipget_mpf)

Acquires a fixed-sized memory block. The only difference between these service calls and get_mpf or tget_mpf is that an error code is immediately returned and the task does not enter the WAITING state when no memory blocks are available in the memory pool.

(5) Releases Fixed-sized Memory Block (rel_mpf, irel_mpf)

Releases a fixed-sized memory block. When a task is waiting to acquire a memory block, either service call makes the task leave the WAITING state.

(6) Refers to Fixed-sized Memory Pool Status(ref_mpf, iref_mpf)

Refers to the state of a fixed-sized memory pool, including the number of available memory blocks and the IDs of waiting tasks.

3.12.2 Notes

The kernel generates management tables in the fixed-sized memory pool area. If the management table is destroyed, correct system operation cannot be guaranteed .

3.13 Variable-Sized Memory Pool

3.13.1 Functions

The variable-sized memory pool is object that dynamically allocates and deallocates memory blocks of any size.

Compared to the fixed-sized memory pool, the variable-sized memory pool has the advantage that it can allocate memory blocks of any size, but there is a drawback to it in that it has large memory-acquiring/returning overhead. Furthermore, there is the problem associated with memory fragmentation, as will be described later.

When creating a variable-sized memory pool using the cfg file, be sure to specify the area in which to create the memory pool and its allocatable maximum size.

There are following method to create a variable-sized memory pool. Note, the wait queue for acquiring memory block is managed by FIFO order.

- Static creation
  Describe "variable_memorypool[]" statement in the cfg file.
- Dynamic creation
  Use cre_mpl or acre_mpl service call.

Following information are specified at creating a variable -sized memory pool.

- Memory pool size
- Maximum size of the memory block acquirable
- Start address of memory pool area for dynamic creation, or section name of the memory pool area for static creation

The kernel offers the following variable-sized memory pool function service calls.

(1) Creates Variable-sized Memory Pool (cre_mpl, acre_mpl)

The cre_mpl creates a variable-sized memory pool with specified variable-sized memory pool ID. The acre_mpl creates a variable-sized memory pool and return the variable-sized memory pool ID.

The variable-sized memory pool area should be the inside of memory object which can be accessed by tasks that use memory blocks. Two or more variable-sized/fixed-sized memory pools may be put in the same memory object. Note, different access permission at every memory block cannot be set.

The cre_mpl and acre_mpl can be issued from the task that belongs to the trusted domain.

(2) Deletes Variable-sized Memory Pool (del_mpl)

The del_mpl deletes a variable-sized memory pool with specified variable-sized memory pool ID. The deleted variable-sized memory pool ID can be used by cre_mpl or acre_mpl.

The del_mplcan be issued from the task that belongs to the trusted domain.

(3) Gets Variable-sized Memory Block (get_mpl, tget_mpl)

Acquires a variable-sized memory block. When the variable-sized memory pool lacks the space for allocation of the memory block, the task enters the WAITING state until the memory pool has enough available space.

(4) Gets Variable-sized Memory Block (Polling) (pget_mpl, ipget_mpl)

Acquires a variable-sized memory block. The only difference between these service calls and get_mpl or tget_mpl is that an error is immediately returned and the task does not enter the WAITING state when no memory block can be acquired from the memory pool.

(5) Releases Variable-sized Memory Block (rel_mpl)

Releases a variable-sized memory block.

Releasing a memory block increases the amount of available space in the variable-sized memory pool. When a task has been waiting to acquire a block and the release of another block gives the memory pool enough available space, the task leaves the WAITING state and acquires the requested memory block.

(6) Refers to Variable-sized Memory Pool Status (ref_mpl, iref_mpl)

Refers to the state of a variable-sized memory pool, including the total amount of available memory, the maximum size of a contiguous memory block, and the IDs of waiting tasks.

3.13.2 About the Fragmentation of Free Spaces

As memory blocks are repeatedly acquired and freed (returned) from the variable-sized memory pool, the free spaces of memory will become fragmented, resulting in a situation where the total size of free spaces is adequate, but there are no contiguous free spaces, or a situation where it is impossible to acquire a large memory block (Figure 3.15).

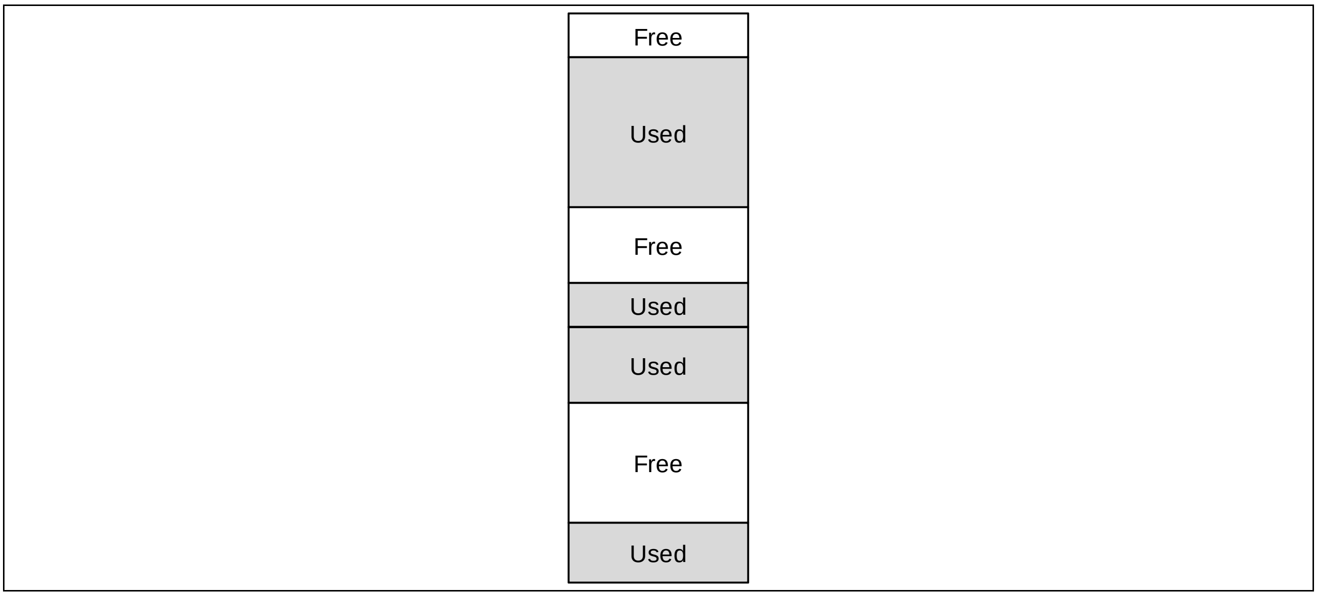

Figure 3.15 Fragmentation of Free Space

In this kernel, variations of memory block sizes are limited to make the memory less liable to become fragmented.

Variations of memory block sizes are determined based on variable_memorypool[].max_memsize in the cfg file. For details, refer to section 8.4.15, "Variable-sized Memory Pool Definition (variable_memorypool[])."

3.13.3 Notes

The kernel generates management tables in the variable-sized memory pool area. If the management table is destroyed, correct system operation cannot be guaranteed .

3.14 Time Management Function

The kernel provides the following functions related to time management:

- Reference to and setting of the system clock
- Time event handler (cyclic handler and alarm handler) execution control
- Task execution control such as timeout

The unit of time used to define time parameters for the service calls is millisecond.

3.14.1 Task Timeout

Timeout values for WAITING states are specifiable with service calls that start with t, such as tslp_tsk and twai_sem.

When the wait condition has not been satisfied after the specified timeout period has elapsed, the task is taken out of the WAITING state and the error code E_TMOUT is returned as the return value for the service call.

Timeouts can be used to detect abnormal behavior in the form of events that should have been generated within the timeout period but were not.

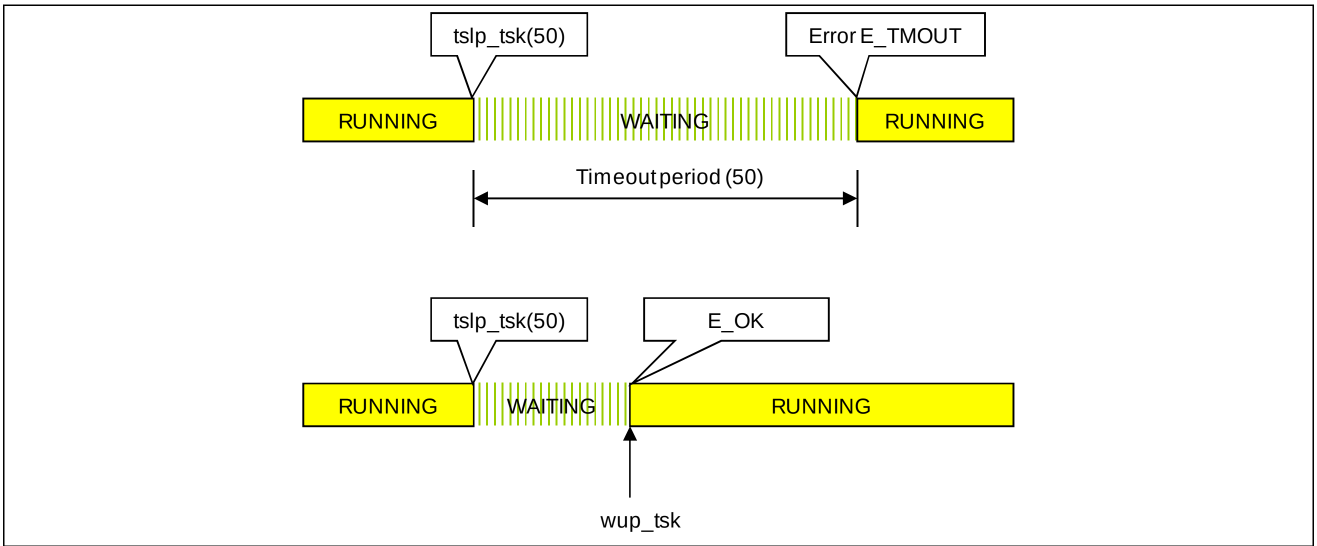

Figure 3.16 Timeout

3.14.2 Task Delay

Using dly_tsk, it is possible to place a task into a wait state for a specified duration of time. When the specified time elapses, the task is released from the wait state and E_OK is returned.

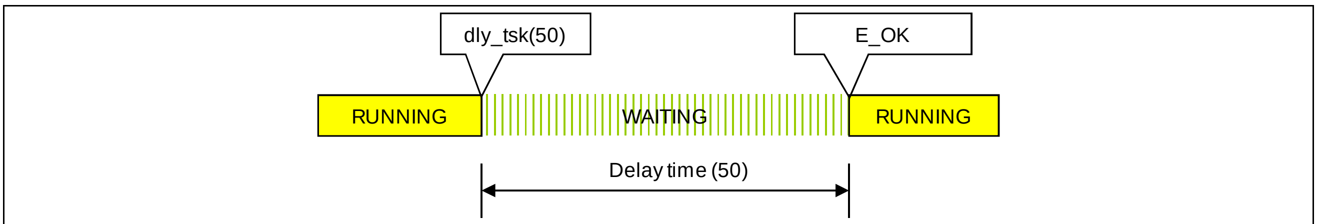

Figure 3.17 Task Delay

3.14.3 Cyclic Handler

The cyclic handler is time event handler that is activated in each activation cycle after a specified activation phase has elapsed. There are two methods to activate the cyclic handler, in one of which the activation phase is saved and in the other the activation phase is not saved. In the former case, the cyclic handler activation time is determined relative to the time at which the cyclic handler was created (system startup time). In the latter case, the cyclic handler activation time is determined relative to the time at which the cyclic handler operation is started.

The cyclic handler has passed to it the extended information that was specified when it was created.

There are following method to create a cyclic handler.

- Static creation
  Describe "cyclic_hand[]" statement in the cfg file.
- Dynamic creation
  Use cre_cyc or acre_cyc service call.

Following information are specified at creating a cyclic handler.

- Cyclic handler start address
- Activation cycle
- Activation phase
- Whether to start operation (TA_STA) or not
- Whether to save the phase (TA_PHS) or not
- Extended information

Figure 3.18 and Figure 3.19 show examples of how the cyclic handler will operate.

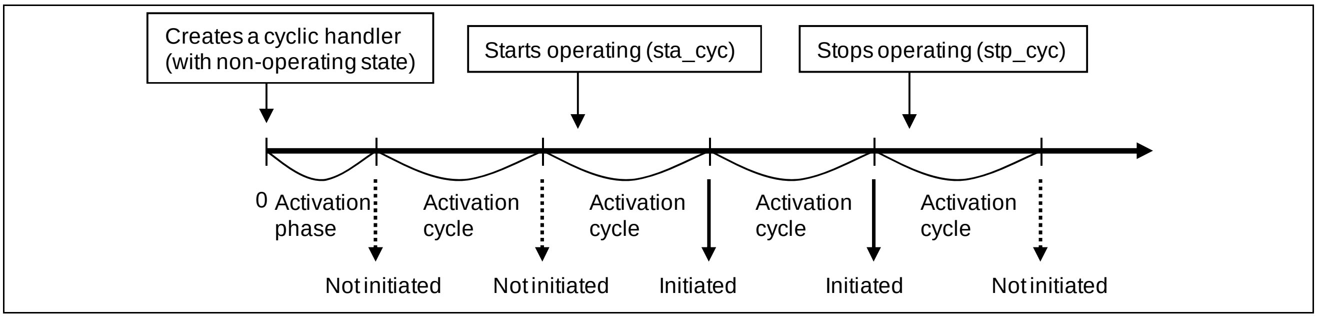

Figure 3.18 Examples of Cyclic Handler Operation (Save phase)

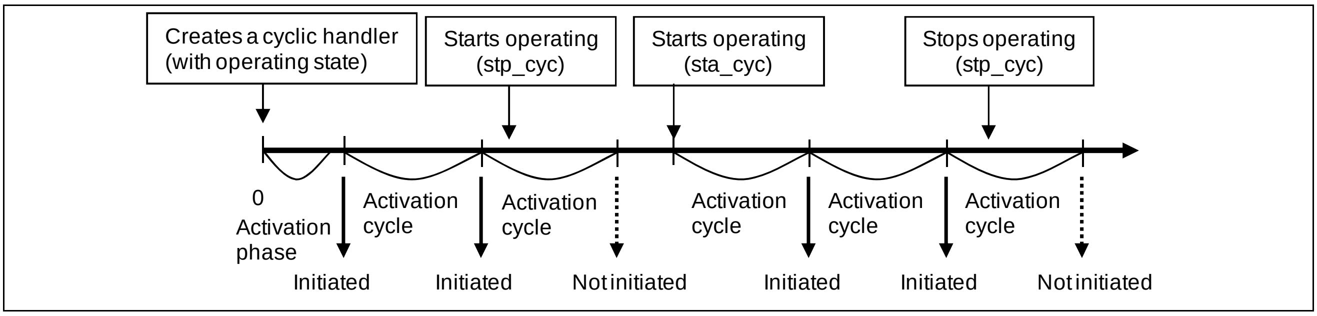

Figure 3.19 Examples of Cyclic Handler Operation (Not save phase)

The kernel offers the following cyclic handler function service calls.

(1) Creates Cyclic Handler (cre_cyc, acre_cyc)

The cre_cyc creates a cyclic handler with specified cyclic handler ID. The acre_cyc creates a cyclic handler and return the cyclic handler ID.

The cre_cyc and acre_cyc can be issued from the task that belongs to the trusted domain.

(2) Deletes Cyclic Handler (del_cyc)

The del_cyc deletes a cyclic handler with specified cyclic handler ID. The deleted cyclic handler ID can be used by cre_cyc or acre_cyc.

The del_cyc can be issued from the task that belongs to the trusted domain.

(3) Starts Cyclic Handler Operation (sta_cyc, ista_cyc)

Starts a cyclic handler operation.

(4) Stops Cyclic Handler Operation (stp_cyc, istp_cyc)

Stops a cyclic handler operation.

(5) Refers to Cyclic Handler Status (ref_cyc, iref_cyc)

Refers to the operating state of the cyclic handler, including the time left until the cyclic handler is initiated.

3.14.4 Alarm Handler

The alarm handler is a time event handler that is activated only once when a specified time of day is reached. Using the alarm handler, it is possible to perform processing dependent on the time of day.

Figure 3.20 shows an example of how the alarm handler will operate.

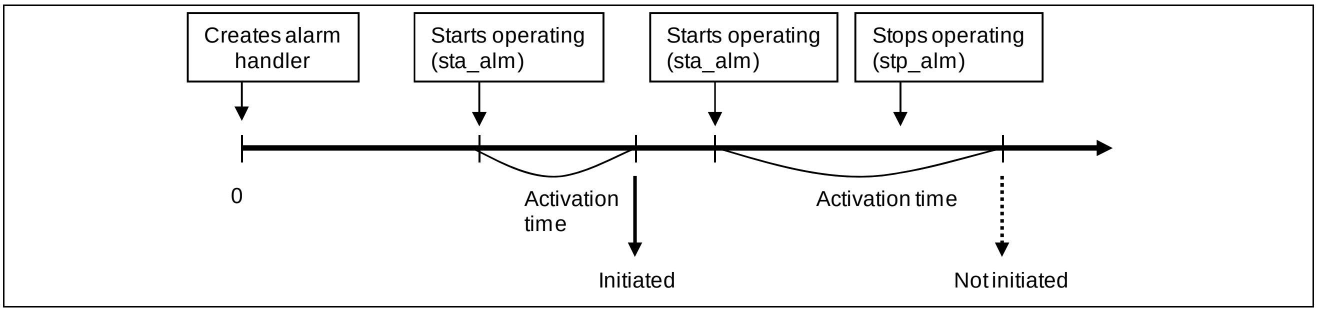

Figure 3.20 Example of Alarm Handler Operation

The alarm handler has passed to it the extended information that was specified when it was created.

There are following method to create a alarm handler.

- Static creation
  Describe "alarm_hand[]" statement in the cfg file.
- Dynamic creation
  Use cre_alm or acre_alm service call.

Following information are specified at creating a alarm handler.

- Alarm handler start address
- Extended Information

The kernel offers the following alarm handler function service calls.

(1) Creates Alarm Handler (cre_alm, acre_alm)

The cre_alm creates a alarm handler with specified alarm handler ID. The acre_alm creates a alarm handler and return the alarm handler ID.

The cre_alm and acre_alm can be issued from the task that belongs to the trusted domain.

(2) Deletes Alarm Handler (del_alm)

The del_alm deletes a alarm handler with specified alarm handler ID. The deleted alarm handler ID can be used by cre_alm or acre_alm.

The del_alm can be issued from the task that belongs to the trusted domain.

(3) Starts Alarm Handler Operation (sta_alm, ista_alm)

Initiates an alarm handler after the specified time has elapsed.

(4) Stops Alarm Handler Operation (stp_alm, istp_alm)

Stops an alarm handler operation.

(5) Refers to Alarm Handler Status (ref_alm, iref_alm)

Refers to the operating status of the alarm handler and the time left until the alarm handler is initiated.

3.14.5 Accuracy of the Time

All of time-out and other time parameters are in millisecond units.

The accuracy of these items of time is TIC_NUME / TIC_DENO [ms]. Updating of the system time and the management of time are performed with this accuracy. Note that TIC_NUME and TIC_DENO are defined in system.tic_nume and system.tic_deno of the cfg file, respectively.

It is so designed that a time event (e.g., time-out occurrence or cyclic handler activation) will not occur before a specified relative time elapses.

Figure 3.21 shows examples where tslp_tsk(5) is executed at 9.2 [ms] in actual time.

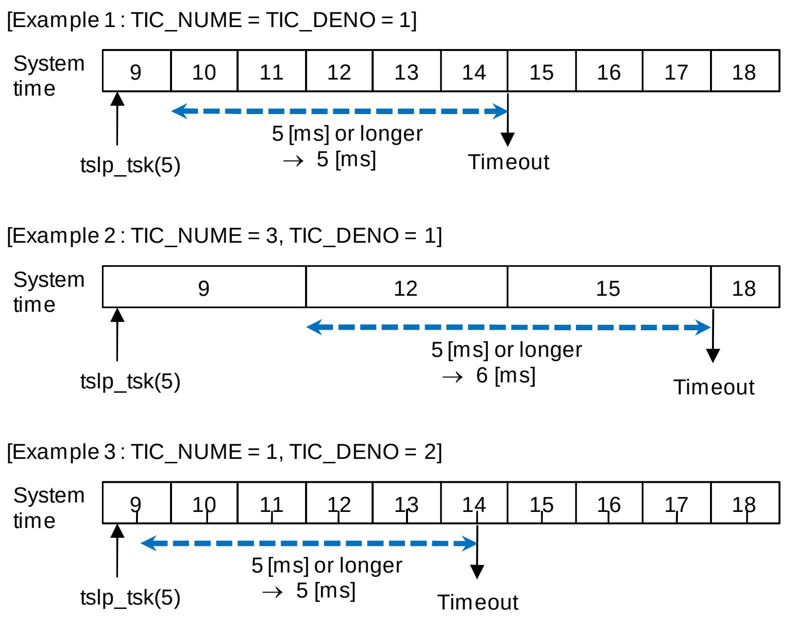

Figure 3.21 Accuracy of the Time (tslp_tsk)

For cyclic handlers, the relative time in each occurrence is handled as described below.

(1) Cyclic handlers without TA_PHS attribute
- (a) When operation starts in sta_cyc or ista_cyc
  The relative time in the n'th occurrence, referenced to the sta_cyc or ista_cyc point of time, is handled as the value derived by the equation below.
  (activation cycle) × n
- (b) When operation starts after the TA_STA attribute is specified in the cfg file when the handler is generated
  The relative time in the n'th occurrence, referenced to the system startup point of time (handler generation point of time), is handled as the value derived by the equation below.
  (activation phase) + (activation cycle) × (n - 1)

(2) Cyclic handlers with TA_PHS attribute
(Handled the same way as in (b) of (1). However, whether the handler is actually activated depends on the handler's operating status.

3.14.6 Precautions

When a timer interrupt is generated, the kernel performs the processing described below.

(a) Updates the system time
(b) Activates and runs alarm handler
(c) Activates and runs cyclic handler
(d) Processes task timeout processing specified by service calls with the timeout function and dly_tsk

These items of processing are all executed while the interrupts whose priority levels are below that of the timer interrupt (clock.IPL) are masked.

Of the above, (b), (c) and (d) could involve duplicate processing of multiple tasks or handlers, in which case the kernel's processing time may be extended by a huge amount of time. This will bring about the following adverse effects:

- Deteriorated responses to interrupts
- Delay in the system time

To avoid these, strictly observe the following:

- Reduce time event handler processing to the shortest time possible.
- Use as large values as possible for time event handler cycles and the time-out time specified in service calls with time-outs included. Think of an extreme case where a cyclic handler is assigned a cycle time of 1 ms, for example, and processing of the handler requires 1 ms or more. In such case, none but the cyclic hander alone will be executed forever, causing in effect the system to hang.

3.15 System State Management Function

(1) Rotates Task Precedence (rot_rdq, irot_rdq)

This service call establishes the time-sharing system (TSS). That is, rotating the ready queue at regular intervals accomplishes the round-robin scheduling required for the TSS.

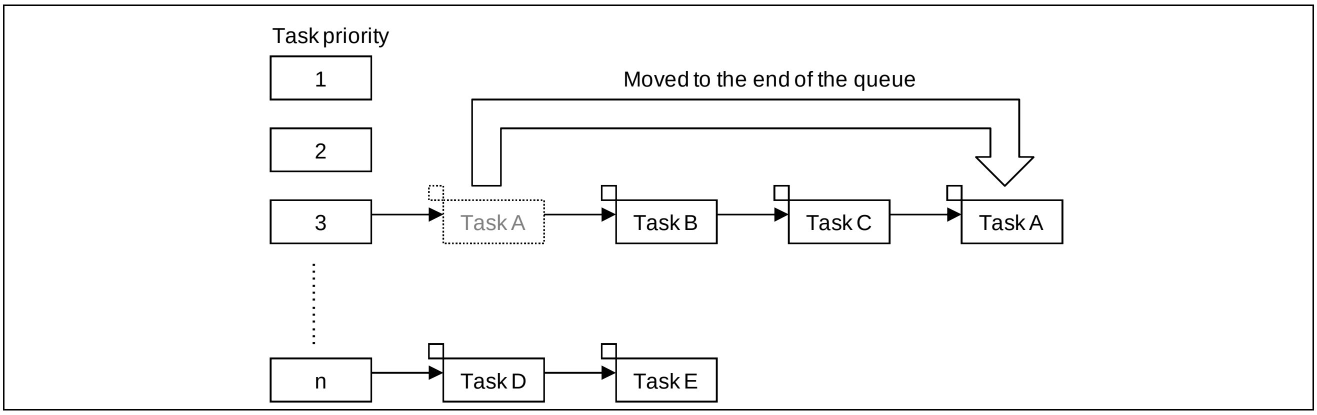

Figure 3.22 Operation of the Ready Queue by rot_rdq

(2) Refers to Task ID in the RUNNING State (get_tid, iget_tid)

Refers to the ID of the task in the RUNNING state. When iget_tid is issued in a non-task context, the ID of the task running at the time is referred.

(3) Locks the CPU (loc_cpu, iloc_cpu) and Unlocks the CPU (unl_cpu, iunl_cpu)

The loc_cpu or iloc_cpu makes the system enter the CPU-locked state. To subsequently leave the CPU-locked state, issue unl_cpu or iunl_cpu.

(4) Disables Dispatching (dis_dsp) and Enables Dispatching (ena_dsp)

The dis_dsp makes the system enter the dispatching-disabled state. To subsequently leave the dispatching-disabled state, issue ena_dsp.

(5) Refers to Context State (sns_ctx)

Checks whether the system is in a task or non-task context.

(6) Refers to CPU-Locked State (sns_loc)

Checks whether the system is in the CPU-locked state or not.

(7) Refers to Dispatching-Disabled State (sns_dsp)

Checks whether the system is in the dispatching-disabled state or not.

(8) Refers to Dispatching-Pending State (sns_dpn)

Checks if the system is in the dispatching-pending state.

The dispatching-pending state means that processing with a higher priority than the dispatcher is in progress so that no other task can be executed. To be more specific, each of the following cases corresponds to the dispatching-pending state.

- CPU-locked state
- Dispatching-disabled state
- Non-task context

Unless the system is in the dispatch-pending state, all service calls to make a task enter the WAITING state are available. When developing software (e.g., middleware) that may be invoked from any system state, this service call (sns_dpn) is useful for checking the current system state to see whether a service call that makes a task enter the WAITING state can be processed or will lead to an error being returned.

(9) Starts Kernel (vsta_knl, ivsta_knl)

Initiates the kernel according to the results of configuration.

(10) System Down (vsys_dwn, ivsys_dwn)

Makes the system go down and initiates the system-down routine.

3.16 Interrupt Management Function

When an interrupt is generated, the corresponding interrupt handler is initiated. Interrupt handlers should be defined through interrupt_vector[] (for relocatable-vector) or interrupt_fvector[] (for fixed-vector).

Also refer to section 2.7, "Interrupts."

(1) Changes Interrupt Mask (chg_ims, ichg_ims)

Changes the interrupt mask (IPL bits in the PSW register) to the specified value.

In task-context, the system enters to dispatching-disabled state when the IPL bit is changed to other than 0, and the system enters to dispatching-enabled state when the IPL bit is changed to 0.

(2) Refers to Interrupt Mask (get_ims, iget_ims)

Refers to the interrupt mask.

(3) Returns from Kernel Interrupt Handler (ret_int)

Returns from a kernel interrupt handler.

The user don't have to describe calling ret_int because the code calls ret_int at the end of the interrupt handler is generated with the configuration.

3.17 System Configuration Management Function

(1) Refers to Version Information (ref_ver, iref_ver)

Refers to the version numbers of the kernel and the µITRON specification implemented in the kernel. The information acquired by ref_ver or iref_ver can also be acquired through kernel configuration macros (refer to section 4.2.1, "Constant Macro").

3.18 Object Reset Function

The object reset function initializes specified object.

This is the function outside μITRON 4.0 specification.

(1) Resets Data Queue (vrst_dtq)

Resets a data queue. Tasks in waiting to send data are released from the WAITING state, and error EV_RST is returned to the tasks. And data stored in the data queue is annulled.

(2) Resets Mailbox (vrst_mbx)

Resets a mailbox. Messages queued in the mailbox comes off from the management of the kernel.

(3) Resets Message Buffer (vrst_mbf)

Resets a message buffer. Tasks in waiting to send a message are released from the WAITING state, and error EV_RST is returned to the tasks. And messages stored in the message buffer is annulled.

(4) Resets Fixed-sized Memory Pool (vrst_mpf)

Resets a fixed-sized memory pool. Tasks in waiting to acquire a memory block are released from the WAITING state, and error EV_RST is returned to the tasks.

And memory blocks which has been allocated are handled as free space. Therefore, application program must not access the memory blocks which has been acquired after issuing this service call.

(5) Resets Variable-sized Memory Pool (vrst_mpl)

Resets a variable-sized memory pool. Tasks in waiting to acquire a memory block are released from the WAITING state, and error EV_RST is returned to the tasks.

And memory blocks which has been allocated are handled as free space. Therefore, application program must not access the memory blocks which has been acquired after issuing this service call.

3.19 Memory Object Management Function

The memory object management functions are used to perform memory object operations such as register, unregister, change permission, check permission, etc.

There are following method to register a memory object.

- Static registration
  Describe "memory_object[]" statement in the cfg file.
- Dynamic registration
  Use ata_mem service call.

Following information are specified at registration a memory object.

- Start address of memory object
- Size of memory object for dynamic registration, or termination address of memory object for static registration
- Access permission vector

The kernel offers the following memory object management function service calls.

(1) Registers Memory Object (ata_mem)

The ata_mem registers the specified memory area as memory object with specified access permission.

Note, the error E_OACV might return by the restriction of number of memory objects. For details, refer to section 2.9.3, "Restriction in the Number of Memory Objects."

The ata_mem can be issued from the task that belongs to the trusted domain.

(2) Unregisters Memory Object (det_mem)

The det_mem unregisters specified memory object.

The det_mem can be issued from the task that belongs to the trusted domain.

(3) Changes Access Permission (sac_mem)

The sac_mem changes access permission for specified memory object.

Note, the error E_OACV might return by the restriction of number of memory objects. For details, refer to section 2.9.3, "Restriction in the Number of Memory Objects."

The sac_mem can be issued from the task that belongs to the trusted domain.

(4) Checks Access Permission (vprb_mem)

The vprb_mem checks whether specified task can do specified access to specified memory area.

(5) Refers Memory Object State (ref_mem)

The ref_mem refers to access permission vector for specified memory object.

4. Data Macros

This chapter describes the data types and macros, which are used when issuing service calls provided by the RI600/PX.

4.1 Data Types

The Following lists the data types of parameters specified when issuing a service call. Macro definition of the data type is performed by "inc600\itron.h" or "inc600\kernel.h" which is included from "itron.h".

Table 4.1 Data Types

Where	Macro	Data Type	Description
itron.h	B	signed char	Signed 8-bit integer
	H	signed short	Signed 16-bit integer
	W	signed long	Signed 32-bit integer
	D	signed long long	Signed 64-bit integer
	UB	unsigned char	Unsigned 8-bit integer
	UH	unsigned short	Unsigned 16-bit integer
	UW	unsigned long	Unsigned 32-bit integer
	UD	unsigned long long	Unsigned 64-bit integer
	VB	unsigned char	8-bit value with unknown data type
	VH	unsigned short	16-bit value with unknown data type
	VW	unsigned long	32-bit value with unknown data type
	VD	unsigned long long	64-bit value with unknown data type
	VP	void *	Pointer to unknown data type
	FP	void (*)	Processing unit start address (pointer to a function)
	INT	signed long	Signed 32-bit integer
	UINT	unsigned long	Unsigned 32-bit integer
	BOOL	signed long	Boolean value (TRUE or FALSE)
	ER	signed long	Error code
	ID	signed short	Object ID
	ATR	unsigned short	Object attribute
	STAT	unsigned short	Object state
	MODE	unsigned short	Service call operational mode
	PRI	signed short	Priority of task or message
	SIZE	unsigned long	Memory area size (in bytes)
	TMO	signed long	Timeout (in milliseconds)
	RELTIM	unsigned long	Relative time (in milliseconds)
	VP_INT	signed long	Pointer to unknown data type, or signed 32-bit integer
	ER_ID	signed long	Error code, or object ID
	ER_UINT	signed long	Error code, or signed 32-bit integer
	ER_BOOL	signed long	Error code, or boolean value (TRUE or FALSE)
kernel.h	ACPTN	unsigned short	Access permission pattern
	FLGPTN	unsigned long	Eventflag bit pattern
	IMASK	unsigned short	Interrupt mask
	TEXPTN	unsigned long	Task exception code

4.2 Macro

4.2.1 Constant Macro

Table 4.2 Constant Macro

Class	Macro	Definition	Where	Description
General	NULL	0	itron.h	Null pointer
	TRUE	1	itron.h	True
	FALSE	0	itron.h	False
	E_OK	0	itron.h	Normal completion
Object attribute	TA_NULL	0	itron.h	Object attributes unspecified
	TA_HLNG	0x0000	kernel.h	High-level language interface
	TA_ASM	0x0001	kernel.h	Assembly language interface
	TA_TFIFO	0x0000	kernel.h	Task queue in FIFO order
	TA_TPRI	0x0001	kernel.h	Task queue in order of task priority
	TA_MFIFO	0x0000	kernel.h	Message queue in FIFO order
	TA_MPRI	0x0002	kernel.h	Message queue in order of message priority
	TA_ACT	0x0002	kernel.h	Task is activated after creation
	TA_WSGL	0x0000	kernel.h	Do not allow multiple tasks to wait for eventflag
	TA_WMUL	0x0002	kernel.h	Allow multiple tasks to wait for eventflag
	TA_CLR	0x0004	kernel.h	Clear eventflag when freed from WAITING state
	TA_CEILING	0x0003	kernel.h	Priority ceiling protocol
	TA_STA	0x0002	kernel.h	Create cyclic hander in operational state
	TA_PHS	0x0004	kernel.h	Save cyclic hander phase
Timeout	TMO_POL	0	itron.h	Polling
	TMO_FEVR	-1	itron.h	Waiting forever
	TMO_NBLK	-2	itron.h	Non blocking
Operation mode	TWF_ANDW	0x0000	kernel.h	Eventflag AND wait
	TWF_ORW	0x0001	kernel.h	Eventflag OR wait
Task exception	TTEX_ENA	0x0000	kernel.h	Task exception enabled state
	TTEX_DIS	0x0001	kernel.h	Task exception disabled state
State	TTS_RUN	0x0001	kernel.h	RUNNING state
	TTS_RDY	0x0002	kernel.h	READY state
	TTS_WAI	0x0004	kernel.h	WAITING state
	TTS_SUS	0x0008	kernel.h	SUSPENDED state
	TTS_WAS	0x000C	kernel.h	WAITING-SUSPENDED state
	TTS_DMT	0x0010	kernel.h	DORMANT state
	TTW_SLP	0x0001	kernel.h	Sleeping state
	TTW_DLY	0x0002	kernel.h	Delayed state
	TTW_SEM	0x0004	kernel.h	Waiting state for a semaphore resource
	TTW_FLG	0x0008	kernel.h	Waiting state for an eventflag
	TTW_SDTQ	0x0010	kernel.h	Sending waiting state for a data queue
	TTW_RDTQ	0x0020	kernel.h	Receiving waiting state for a data queue
	TTW_MBX	0x0040	kernel.h	Receiving waiting state for a mailbox
	TTW_MTX	0x0080	kernel.h	Waiting state for a mutex
	TTW_SMBF	0x0100	kernel.h	Sending waiting state for a message buffer
	TTW_RMBF	0x0200	kernel.h	Receiving waiting state for a message buffer

Class	Macro	Definition	Where	Description
	TTW_MPF	0x2000	kernel.h	Waiting state for a fixed-sized memory block
	TTW_MPL	0x4000	kernel.h	Waiting state for a variable-sized memory block
	TCYC_STP	0x0000	kernel.h	Cyclic handler in non-operational state
	TCYC_STA	0x0001	kernel.h	Cyclic handler in operational state
	TALM_STP	0x0000	kernel.h	Alarm handler in non-operational state
	TALM_STA	0x0001	kernel.h	Alarm handler in operational state
Other constant	TSK_SELF	0	kernel.h	Specify invoking task
	TSK_NONE	0	kernel.h	No relevant task
	TPRI_SELF	0	kernel.h	Specify base priority of invoking task
	TPRI_INI	0	kernel.h	Specify initial priority
Kernel configuration	TMIN_TPRI	1	kernel.h	Minimum task priority
	TMAX_TPRI	system.priority	kernel_id.h	Maximum task priority
	TMIN_MPRI	1	kernel.h	Minimum message priority
	TMAX_MPRI	system.message_pri	kernel_id.h	maximum message priority
	TKERNEL_MAKER	0x011B	kernel.h	Kernel maker code
	TKERNEL_PRID	0x0004	kernel.h	Identification number of the kernel
	TKERNEL_SPVER	0x5403	kernel.h	Version number of the ITRON specification
	TKERNEL_PRVER	0x0100	kernel.h	Version number of the kernel
	TMAX_ACTCNT	255	kernel.h	Maximum number of queued task activation requests
	TMAX_WUPCNT	255	kernel.h	Maximum number of queued task wakeup requests
	TMAX_SUSCNT	1	kernel.h	Maximum number of nested task suspension requests
	TBIT_FLGPTN	32	kernel.h	Number of bits in an eventflag
	TBIT_TEXPTN	32	kernel.h	Number of bits in task exception code
	TIC_NUME	system.tic_nume	kernel_id.h	Time tick period numerator
	TIC_DENO	system.tic_deno	kernel_id.h	Time tick period denominator
	TMAX_MAXSEM	65535	kernel.h	Maximum value of the maximum semaphore resource count
	VTMAX_DOMAIN	*1	kernel_id.h	Maximum domain ID
	VTMAX_TSK	*1	kernel_id.h	Maximum task ID
	VTMAX_SEM	*1	kernel_id.h	Maximum semaphore ID
	VTMAX_FLG	*1	kernel_id.h	Maximum eventflag ID
	VTMAX_DTQ	*1	kernel_id.h	Maximum data queue ID
	VTMAX_MBX	*1	kernel_id.h	Maximum mailbox ID
	VTMAX_MTX	*1	kernel_id.h	Maximum mutex ID
	VTMAX_MBF	*1	kernel_id.h	Maximum message buffer ID
	VTMAX_MPF	*1	kernel_id.h	Maximum fixed-sized memory pool ID
	VTMAX_MPL	*1	kernel_id.h	Maximum variable-sized memory pool ID
	VTMAX_CYH	*1	kernel_id.h	Maximum cyclic handler ID
	VTMAX_ALH	*1	kernel_id.h	Maximum alarm handler ID
	VTSZ_MBFTBL	4	kernel.h	Size of message buffer's message management table (in bytes)
	VTMAX_AREASIZE	0x10000000	kernel.h	Maximum size of various areas (in bytes)

Class	Macro	Definition	Where	Description
	VTKNL_LVL	system.system _IPL	kernel_id.h	Kernel interrupt mask level
	VTIM_LVL	clock.IPL	kernel_id.h	Timer interrupt priority level
Error code	E_SYS	-5	itron.h	System error
	E_NOSPT	-9	itron.h	Unsupported function
	E_RSFN	-10	itron.h	Reserved function code
	E_RSATR	-11	itron.h	Reserved attribute
	E_PAR	-17	itron.h	Parameter error
	E_ID	-18	itron.h	Invalid ID number
	E_CTX	-25	itron.h	Context error
	E_MACV	-26	itron.h	Memory access violation
	E_OACV	-27	itron.h	Object access violation
	E_ILUSE	-28	itron.h	Illegal use of service call
	E_NOMEM	-33	itron.h	Insufficient memory
	E_NOID	-34	itron.h	No ID number available
	E_OBJ	-41	itron.h	Object state error
	E_NOEXS	-42	itron.h	Non-existent object
	E_QOVR	-43	itron.h	Queuing overflow
	E_RLWAI	-49	itron.h	Forced release from waiting
	E_TMOUT	-50	itron.h	Polling failure or timeout
	E_DLT	-51	itron.h	Waiting object deleted
	E_CLS	-52	itron.h	Waiting object state changed
	EV_RST	-127	itron.h	Released from WAITING state by the object reset
Protection extension	TDOM_SELF	0	kernel.h	Domain that invoking task belongs
	TACP_SHARED	((1u << (VTMAX_DOMAIN)) -1)	kernel.h	Access permission pattern that represents "all domain can access"
	TACT_SRW *2	{TACP_SHARED, TACP_SHARED, TACP_SHARED}	kernel.h	Access permission vector that represents "all types of access (read, write, execute) are permitted for all domains."
	TACT_SRO *2	{TACP_SHARED, 0, TACP_SHARED}	kernel.h	Access permission vector that represents "all domain cannot write, and can read and execute"
	TPM_READ	1	kernel.h	Operand read access
	TPM_WRITE	2	kernel.h	Operand write access
	TPM_EXEC	4	kernel.h	Execution access

Notes: 1 Depends on the result of configuration.

2 This can be describe only at the right of an initial assignment statement.

4.2.2 Function Macro

(1) ER MERCD(ER ercd)

Description	Returns main error code for ercd.	
Header file	itron.h	
Argument	ercd	Error code
Return Value	Main error code for ercd.	

(2) ER SERCD(ER ercd)

Description	Returns sub-error code for ercd.	
Header file	itron.h	
Argument	ercd	ercd
Return Value	Sub-error code for ercd.	
Remarks	Sub-error code of the error code that will be returned from the kernel is -1.	

(3) ER ERCD(ER mercd, ER sercd)

Description	Returns error code consisting of the main error code (mercd) and sub-error code (sercd).	
Header file	itron.h	
Argument	mercd	Main error code
	sercd	Sub-error code
Return Value	Error code	

(4) SIZE TSZ_DTQ(UINT dtqcnt)

Description	Returns the size of a data queue area in which the dtqcnt number of data items can be stored. (in bytes)	
Header file	kernel.h	
Argument	dtqcnt	Number of data items
Return Value	Size of data queue area	

(5) SIZE TSZ_MPF(UINT blkcnt, UINT blksz)

Description	Returns the size of a fixed-sized memory pool from which blkcnt number of blksz-byte memory blocks can be acquired. (in bytes)	
Header file	kernel.h	
Argument	blkcnt	Number of memory blocks
	blksz	Memory block size
Return Value	Size of fixed-sized memory pool	

(6) SIZE TSZ_MPFMB(UINT blkcnt, UINT blksz)

Description	Returns the size of the management area required for a fixed-sized memory pool from which blkcnt number of blksz-byte memory blocks can be acquired. (in bytes)	
Header file	kernel.h	
Argument	blkcnt	Number of memory blocks
	blksz	Memory block size
Return Value	Size of fixed-sized memory pool management area	

(7) ATR TA_DOM(ID domid)

Description	Returns attribute which represents to belong to the domain indicated by domid. This macro is used for bit7-4 of tskatr at task creation.	
Header file	kernel.h	
Argument	domid	Domain ID (TDOM_SELF can be specified.)
Return Value	bit7-4 of tskatr	

(8) ACPTN TACP(ID domid)

Description	Returns access permission pattern that represents "only the domain indicated by domid can access".	
Header file	kernel.h	
Argument	domid	Domain ID (TDOM_SELF cannot be specified.)
Return Value	Access permission pattern	

(9) ACVCT TACT_PRW (ID domid)

Description	Returns access permission vector that represents "all types of access (read, write, execute) are permitted only for the domain indicated by domid".	
Header file	kernel.h	
Argument	domid	Domain ID (TDOM_SELF cannot be specified.)
Return Value	Access permission vector	
Remarks	This can be describe only at the right of an initial assignment statement.	

(10) ACVCT TACT_PRO (ID domid)

Description	Returns access permission vector that represents "wirte-access is not permitted for all domain, read and execute-access are permitted only for the domain indicated by domid".	
Header file	kernel.h	
Argument	domid	Domain ID (TDOM_SELF cannot be specified.)
Return Value	Access permission vector	
Remarks	This can be describe only at the right of an initial assignment statement.	

(11) ACVCT TACT_SRPW (ID domid)

Description	Returns access permission vector that represents "read and execute-access are permitted for all domain, write-access is permitted only for the domain indicated by domid".	
Header file	kernel.h	
Argument	domid	Domain ID (TDOM_SELF cannot be specified.)
Return Value	Access permission vector	
Remarks	This can be describe only at the right of an initial assignment statement.	

5. Service Call Reference

5.1 Header File

In the application source, be sure to include kernel.h supplied by the RI600/PX and kernel_id.h output by cfg600px.

5.2 Service Call Return Values and Error Codes

5.2.1 Summary

For service calls that have a return value, a positive value or 0 (E_OK) means that the call is terminated normally, and a negative value represents error code. Although the value returned upon normal termination differs with each service call, most service calls return only E_OK when they are terminated normally.

However, this does not apply to the service calls that have a BOOL-type return value.

5.2.2 Main Error Codes and Sub-Error Codes

The error code consists of the 8 low-order bits that constitute the main error code and the remaining other high-order bits that constitute sub-error code. The sub-error code in all error codes returned by this kernel is -1.

Note that the standard header itron.h has the following macros defined in it:

- ER MERCD (ER ercd) : Retrieves the main error code from error code
- ER SERCD (ER ercd) : Retrieves sub-code from error code
- ER ERCD (ER mercd, ER sercd) : Generates error code from the main error code and sub-error code

5.3 System Status and Service Calls

Whether a service call can be invoked depends on the system status.

5.3.1 Task Context and Non-Task Context

(1) Service calls beginning with "sns"

The service calls whose names begin with "sns" can be invoked from both task contexts and non-task contexts.

(2) Service calls other than (1)

The service calls that begin with i are non-task contexts only, and others are task contexts only.

Be aware that unless invoked in an enabled state, errors (E_CTX error) may be or may not be detected depending on the service call concerned. For details, check for confirmation the error code column of each service call described later in this manual.

5.3.2 CPU-Locked State

The service calls that can be invoked from the CPU-locked state are limited to those that are listed below. If any other service call is invoked from the CPU-locked state, E_CTX error is detected.

- ext_tsk (released from the CPU-locked state)
- exd_tsk (released from the CPU-locked state)
- loc_cpu, iloc_cpu
- unl_cpu, iunl_cpu
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- vsta_knl, ivsta_knl
- vsys_dwn, ivsys_dwn

5.3.3 Dispatching-Disabled State

If a service call that transitions to a wait state is invoked, E_CTX error is returned.

5.3.4 Non-Kernel Interrupt Handler, etc.

When the PSW.IPL is larger than the kernel interrupt mask level (system.system_IPL), such as non-kernel interrupt handlers, do not call service calls[2]. If calling, the service call returns error E_CTX. In this case, the PSW.IPL temporarily falls on the kernel interrupt mask level. Please use this error only for the purpose of debug.

5.4 Other Than µITRON Specification

The service calls whose names begin with the letter "v," "iv," or "V" as for the vrst_dtq service call conform to this kernel's original specification other than µITRON 4.0 specification.

Furthermore, the following "ixxx_yyy" service calls (whose names begin with the letter "i") are the variations of µITRON 4.0-compliant, task context-only "xxx_yyy" service calls that have been made invocable from non-task contexts and are, therefore, not µITRON 4.0 specification.

ista_tsk, ichg_pri, iget_pri, iref_tsk, iref_tst, isus_tsk, irsm_tsk, ifrsm_tsk, ipol_sem, iref_sem,

iclr_flg, ipol_flg, iref_flg, iprcv_dtq, iref_dtq, isnd_mbx, iprcv_mbx, iref_mbx, ipsnd_mbf,

iref_mbf, ipget_mpf, irel_mpf, iref_mpf, ipget_mpl, iref_mpl, iset_tim, iget_tim, ista_cyc,

istp_cyc, iref_cyc, ista_alm, istp_alm, iref_alm, ichg_ims, iget_ims, iref_ver, vprb_mem

[2] Except chg_ims, ichg_ims, get_ims, iget_ims, vsta_knl, ivsta_knl, vsys_dwn, and ivsys_dwn

5.5 Task Management Function

Table 5.1 shows specifications of the task management function.

Table 5.1 Specifications of the Task Management Function

No.	Item	Content
1	Task ID	1 - VTMAX_TSK *1
2	Task priority	1 - TMAX_TPRI *2
3	Maximum number of queued task activation request	255
4	Extension information (parameters passed to task)	32 bits
5	Task attribute	TA_HLNG : Written in high-level language TA_ACT : Activation attribute

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates maximum task ID.

2 This is the macro output to kernel_id.h by cfg600px, which represents the value specified in system.priority.

Table 5.2 Service Calls for Task Management

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_tsk		Creates task	T		E	D	U	
2	acre_tsk		Creates task (Automatic ID Assignment)	T		E	D	U	
3	del_tsk		Deletes task	T		E	D	U	
4	act_tsk	[S]	Activates task	T		E	D	U	
5	iact_tsk	[S]			N	E	D	U	
6	can_act	[S]	Cancels task activation requests	T		E	D	U	
7	ican_act				N	E	D	U	
8	sta_tsk	[B]	Activates task (with a start code)	T		E	D	U	
9	ista_tsk				N	E	D	U	
10	ext_tsk	[S][B]	Terminates invoking task	T		E	D	U	L
11	exd_tsk		Terminates and deletes invoking task	T		E	D	U	L
12	ter_tsk	[S][B]	Terminates task	T		E	D	U	
13	chg_pri	[S][B]	Changes task priority	T		E	D	U	
14	ichg_pri				N	E	D	U	
15	get_pri	[S]	Refers task priority	T		E	D	U	
16	iget_pri				N	E	D	U	
17	ref_tsk		Refers to task state	T		E	D	U	
18	iref_tsk				N	E	D	U	
19	ref_tst		Refers to task state (simple version)	T		E	D	U	
20	iref_tst				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.5.1 Creates Task (cre_tsk, acre_tsk)

❑C Language API

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

❑Parameter

```
tskid    Task ID
pk_ctsk  Pointer to the packet containing the task creation information
```

❑Packet Structure

```
typedef  struct t_ctsk {
         ATR     tskatr;          Task attribute
         VP_INT  exinf;           Extended information
         FP      task;            Task start address
         PRI     itskpri;         Task initial priority
         SIZE    stksz;           User stack size (in bytes)
         VP      stk;             Start address of user stack
} T_CTSK;
```

❑Return Value

In cre_tsk : E_OK for normal completion or error code
In acre_tsk : Created task ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute (1) Either bit0,bit2,bit3 or from bit8 to bit15 of tskatr is 1. (2) VTMAX_DOMAIN < (Value from bit4 to bit7 of tskatr)
E_PAR	Parameter error (1) pk_ctsk == NULL (2) task == NULL (3) itskpri < 0, TMAX_TPRI < itskpri (4) stk is not 16-bytes boundary. (5) stksz is not multiple of 16. (6) stksz < (minimum size decided by system.context),[3] VTMAX_AREASIZE < stksz (7) stk + stksz > 0x100000000,
E_ID	Invalid ID number (only for cre_tsk) tskid<=0, VTMAX_TSK < tskid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_ctsk has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOMEM	Insufficient memory stk == NULL
E_NOID	No ID number available (only for acre_tsk)
E_OBJ	Object state error (only for cre_tsk) The task indicated by tskid exists.

[3] Refer to.Table 11.2

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_tsk creates a task with task ID indicated by tskid according to the content of pk_ctsk. The acre_tsk creates a task according to the content of pk_ctsk, and returns the created task ID.

The processing performed at task creation is shown in Table 5.3.

Table 5.3 Processing Performed at Task Creation

No.	Content of processing
1	Clears the number of queued activation requests.
2	Resets the task state so that the task exception handling routine is not defined.

(1) Task ID (tskid)

The cre_tsk creates a task with the task ID indicated by tskid.

(2) Task Attribute (tskatr)

The following are specified for tskatr.

```
tskatr:= ( TA_HLNG | [TA_ACT] | [TA_DOM(domid)] ).
```

The bit position of tskatr is shown as follows.

bit15 ~ bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	Domain ID (TA_DOM(domid))				0	0	TA_ACT (= 2)	TA_HLNG (= 0)

- TA_HLNG (0x0000)
  Only C-language is supported for task description language.

- TA_ACT (0x0002)
  When the TA_ACT attribute is specified, the created task makes a transition to the READY state. The processing performed at task activation is shown in Table 5.4.

Table 5.4 Processing Performed at Task Activation

No.	Content of processing
1	Initializes the task's base priority and current priority.
2	Clears the number of queued wakeup requests.
3	Clears the number of nested suspension count
4	Clears pending exception code
5	Disables task exception

- TA_DOM(domid) (from bit 4 to 7)
  The domain ID that the created task belong to is specified. When 0 is specified, the created task belongs to the domain that the invoking task belong to.
  The domain ID should be specified for argument domid of TA_DOM().When TDOM_SELF is specified for domid, the created task belongs to the domain that the invoking task belong to.

(3) Extended Information (exinf)

When the task is activated by TA_ACT attribute, act_tsk or iact_tsk, exinf is passed to the task as argument. And exinf is passed to the task exception handling routine. The exinf can be widely used by the user, for example, to set information concerning the task.

(4) Task Start Address (task)

Specify the task start address.

(5) Task Initial Priority (itskpri)

Specify initial priority of the task. Ranges of the value that can be specified are from 1 to TMAX_TPRI.

(6) User stack size (stksz), Start address of user stack (stk)

The application acquires user stack area, and specifies the start address for stk and the size for stksz.

Note, the µITRON 4.0 specification defines the function that the kernel allocates user stack area when NULL is specified for stk. But RI600/PX does not support this function.

The user stack area must satisfy the following.

1. The start address must be 16-bytes boundary. If not, error E_PAR is returned.
2. The size must be multiple of 16. If not, error E_PAR is returned.
3. The user stack area must not overwrap with either all user stacks and all other memory objects. If not, an error is not detected and correct system operation cannot be guaranteed.

5.5.2 Deletes Task (del_tsk)

❑C Language API

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

❑Parameter

tskid	Task ID

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number tskid<=0, VTMAX_TSK < tskid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_OBJ	Object state error The task indicated by tskid is not in the DORMANT state.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the task indicated by tskid.

5.5.3 Activates Task (act_tsk, iact_tsk)

❑C Language API

```
ER        act_tsk(ID tskid);
ER        iact_tsk(ID tskid);
```

❑Parameter

```
tskid     Task ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID               Invalid ID number
                   (1) tskid<0, VTMAX_TSK < tskid
                   (2) In an invocations from non-task context, tskid == 0
E_CTX              Context error (invoked from system state not permitted)
E_MACV             Memory access violation (only for act_tsk)
                   Stack pointer points out of user stack for invoking task.
E_NOEXS            Non-existent object
                   The task indicated by tskid does not exist.
E_QOVR             Queuing overflow
                   Activation request count has already reached the maximum value.
```

❑Function

Activates the task indicated by tskid. The activated task transitions from the DORMANT state to the READY state. The processing performed at task activation is shown in Table 5.4

In act_tsk, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

The extended information, which is specified at creation, is passed to the target task.

When the task is not in the DORMANT state, up to 255 task activation requests from service calls act_tsk and iact_tsk can be queued.

5.5.4 Cancels Task Activation Request(can_act, ican_act)

❑C Language API

```
ER_UINT    can_act(ID tskid);
ER_UINT    ican_act(ID tskid);
```

❑Parameter

```
tskid      Task ID
```

❑Return Value

```
Activation request count (positive value or 0), or error code
```

❑Error Code

```
E_ID                Invalid ID number
                    (1) tskid<0, VTMAX_TSK < tskid
                    (2) In an invocations from non-task context, tskid == 0
E_CTX               Context error (invoked from system state not permitted)
                      Note : The E_CTX is not detected in the following cases.
                          (1) Invocation of can_act from non-task context
                          (2) Invocation of ican_act from task context
E_NOEXS             Non-existent object
                    The task indicated by tskid does not exist.
```

❑Function

Gets the number of activation requests that are queued for the task indicated by tskid and returns the result as return parameter, at the same time invalidating all of those activation requests.

In can_act, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

This service call can be invoked for a task in the DORMANT state as the target. In that case, its return value is 0.

5.5.5 Activates Task (with a Start Code) (sta_tsk, ista_tsk)

❑C Language API

```
ER        sta_tsk(ID tskid, VP_INT stacd);
ER        ista_tsk(ID tskid, VP_INT stacd);
```

❑Parameter

tskid	Task ID
stacd	start code

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number tskid<=0, VTMAX_TSK < tskid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for sta_tsk) Stack pointer points out of user stack for invoking task.
E_OBJ	Object state error The task indicated by tskid is not in the DORMANT state.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.

❑Function

Activates the task indicated by tskid. The activated task transitions from the DORMANT state to the READY state. The processing performed at task activation is shown in Table 5.4

The start code specified by stacd is passed to the target task.

5.5.6 Terminates Invoking Task (ext_tsk)

❑C Language API

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

❑Parameter

None

❑Return Value

```
Service calls ext_tsk does not return to the position where it was issued.
When the following error is detected, the system will go down.
E_CTX            Context error (invoked from system state not permitted)
```

❑Function

The ext_tsk service call terminates the invoking task normally. The task state changes from the RUNNING state to the DORMANT state. If activation requests are queued, the invoking task is temporarily terminated and then restarted. The processing performed at restart time is shown in Table 5.4.

The processing performed at task termination time is shown in Table 5.5.

Table 5.5 Processing Performed at Task Termination

No.	Content of processing
1	Unlocks the mutex that is locked by the task.

This service call does not have the function to automatically free the resources except the mutex hitherto occupied by the task (e.g., semaphores and memory blocks). Make sure the task frees these resources before it terminates.

This service call can be invoked from the dispatching-disabled or the CPU-locked state. In that case, the dispatching-disabled or the CPU-locked state is canceled.

Note that when the task returns from the entry function, the same operation as for service call ext_tsk will be performed.

If this service call is invoked from non-task context or non-kernel interrupt handlers, an unrecoverable error is assumed and control jumps to the system-down routine.

5.5.7 Terminates and Deletes Invoking Task (exd_tsk)

❑C Language API

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

❑Parameter

None

❑Return Value

Service calls exd_tsk does not return to the position where it was issued.
When the following error is detected, the system will go down.

E_CTX	Context error (invoked from system state not permitted)

❑Function

The exd_tsk service call terminates the invoking task normally and deletes the task.

The processing performed at task termination time is shown in Table 5.5.

This service call does not have the function to automatically free the resources except the mutex hitherto occupied by the task (e.g., semaphores and memory blocks). Make sure the task frees these resources before it terminates.

This service call can be invoked from the dispatching-disabled or the CPU-locked state. In that case, the dispatching-disabled or the CPU-locked state is canceled.

If this service call is invoked from non-task context or non-kernel interrupt handlers, an unrecoverable error is assumed and control jumps to the system-down routine.

5.5.8 Terminates Task (ter_tsk)

❑C Language API

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

❑Parameter

```
tskid     Task ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

E_ID	Invalid ID number tskid<=0, VTMAX_TSK < tskid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_ILUSE	Illegal use of service call The task indicated by tskid is invoking task.
E_OBJ	Object state error The task indicated by tskid is in the DORMANT state.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.

❑Function

Forcibly terminates the other task indicated by tskid. The other task thus terminated transitions to the DORMANT state. At this time, the processing shown in Table 5.5 is performed.

If any activation request is queued, this service call performs the processing that needs to be executed at task activation time, as shown in Table 5.4, and places the target task into the READY state.

When the task waiting at the top of a message buffer send queue or a variable-sized memory pool memory acquisition queue is forcibly dequeued, it is possible that other tasks (waiting for message buffer transmission or variable-sized memory pool memory acquisition) will be released from the WAITING state.

This service call does not have the function to automatically free the resources except the mutex hitherto occupied by the task (e.g., semaphores and memory blocks). Make sure the task frees these resources before it terminates.

5.5.9 Changes Task Priority (chg_pri, ichg_pri)

❑C Language API

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

❑Parameter

```
tskid     Task ID
tskpri    New base priority of the task
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR            Parameter error
                 tskpri < 0, TMAX_TPRI< tskpri
E_ID             Invalid ID number
                 (1)t skid<0, VTMAX_TSK < tskid
                 (2) In an invocations from non-task context, tskid == 0
E_CTX            Context error (invoked from system state not permitted)
E_MACV           Memory access violation (only for chg_pri)
                 Stack pointer points out of user stack for invoking task.
E_ILUSE          Illegal use of service call
                 Ceiling priority is exceeded.
E_OBJ            Object state error
                 The task indicated by tskid is in the DORMANT state.
E_NOEXS          Non-existent object
                 The task indicated by tskid does not exist.
```

❑Function

Changes the base priority of the task indicated by tskid to the value indicated by tskpri.

In chg_pri, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

Specifying tskpri = TPRI_INI (= 0) causes the task's base priority to be reverted to its initial task priority that was specified when it was created.

The changed task base priority remains effective until the task terminates or this service call is invoked again. When the task goes to the DORMANT state, the task base priority it had before it terminated becomes null. The next time the task is activated, it assumes the initial task priority that was specified when it was created.

Also Changes the current priority of the task indicated by tskid to the value indicated by tskpri. But, the current priority is not changed when the target task has locked mutexes with TA_CEILING attribute.

If the target task has locked mutexes with TA_CEILING attribute or is waiting for mutex to be locked and if the task's base priority specified in tskpri is higher than the ceiling priority of the mutex, E_ILUSE is returned.

5.5.10 Refers to Task Priority (get_pri, iget_pri)

❑C Language API

```
ER        get_pri(ID tskid, PRI *p_tskpri);
ER        iget_pri(ID tskid, PRI *p_tskpri);
```

❑Parameter

tskid	Task ID
p_tskpri	Pointer to storage to which the current priority of the target task is returned

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error p_tskpri == NULL
E_ID	Invalid ID number (1) tskid<0, VTMAX_TSK < tskid (2) In an invocations from non-task context, tskid == 0
E_CTX	Context error (invoked from system state not permitted) Note : The E_CTX is not detected in the following cases. (1) Invocation of get_pri from non-task context (2) Invocation of iget_pri from task context
E_MACV	Memory access violation (only for get_pri) The operand write access to the area indicated by p_tskpri has not been permitted to the invoking task.
E_OBJ	Object state error The task indicated by tskid is in the DORMANT state.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.

❑Function

Gets the current priority of the task indicated by tskid and returns it to the area pointed to by p_tskpri.

In get_pri, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

5.5.11 Refers to Task State (ref_tsk, iref_tsk)

❑C Language API

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

❑Parameter

```
tskid     Task ID
pk_rtsk   Pointer to the storage to which the task state is returned
```

❑Packet Structure

```
typedef   struct t_rtsk {
          STAT   tskstat;    Task state
          PRI    tskpri;     Task current priority
          PRI    tskbpri;    Task base priority
          STAT   tskwait;    Reason for waiting
          ID     wobjid;     Object ID for which the task is waiting
          TMO    lefttmo;    Remaining time until timeout
          UINT   actcnt;     Activation request count
          UINT   wupcnt;     Wakeup request count
          UINT   suscnt;     Suspension count
} T_RTSK;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

```
E_PAR            Parameter error
                 pk_rtsk == NULL
E_ID             Invalid ID number
                 (1) tskid<0, VTMAX_TSK < tskid
                 (2) In an invocations from non-task context, tskid == 0
E_CTX            Context error (invoked from system state not permitted)
                   Note : The E_CTX is not detected in the following cases.
                       (1) Invocation of ref_tsk from non-task context
                       (2) Invocation of iref_tsk from task context
E_MACV           Memory access violation (only for ref_tsk)
                 The operand write access to the area indicated by pk_rtsk has not been
                 permitted to the invoking task.
E_NOEXS          Non-existent object
                 The task indicated by tskid does not exist.
```

❑Function

Refers to the state of the task indicated by tskid and returns it to the area pointed to by pk_rtsk.

In ref_tsk, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

The area pointed to by pk_rtsk has one of the following values returned to it. Note that the data marked by an asterisk "*" is indeterminate when the task is in the DORMANT state.

(1) Task state (tskstat)

Current status of the task. One of the following values is retuned in tskstat.

- TTS_RUN (0x0001) : RUNNING state
- TTS_RDY (0x0002) : READY state
- TTS_WAI (0x0004) : WAITING state
- TTS_SUS (0x0008) : SUSPENDED state
- TTS_WAS (0x000c) : WAITING-SUSPENDED state
- TTS_DMT (0x0010) : DORMANT state

(2) Task current priority (tskpri) *

Current priority of the task.

(3) Task base priority (tskbpri) *

Base priority of the task.

(4) Reason for waiting (tskwait) *

Effective when tskstat = TTS_WAI or TTS_WAS, in which case one of the following values is returned.

- TTW_SLP (0x0001) : WAITING state caused by slp_tsk or tslp_tsk
- TTW_DLY (0x0002) : WAITING state caused by dly_tsk
- TTW_SEM (0x0004) : WAITING state caused by wai_sem or twai_sem
- TTW_FLG (0x0008) : WAITING state caused by wai_flg or twai_flg
- TTW_SDTQ (0x0010) : WAITING state caused by snd_dtq or tsnd_dtq
- TTW_RDTQ (0x0020) : WAITING state caused by rcv_dtq or trcv_dtq
- TTW_MBX (0x0040) : WAITING state caused by rcv_mbx or trcv_mbx
- TTW_MTX (0x0080) : WAITING state caused by loc_mtx or tloc_mtx
- TTW_SMBF (0x0100) : WAITING state caused by snd_mbf or tsnd_mbf
- TTW_RMBF (0x0200) : WAITING state caused by rcv_mbf or trcv_mbf
- TTW_MPF (0x2000) : WAITING state caused by get_mpf or tget_mpf
- TTW_MPL (0x4000) : WAITING state caused by get_mpl or tget_mpl

(5) Object ID for which the task is waiting (wobjid) *

Effective when tskstat = TTS_WAI or TTS_WAS, in which case the target object ID waited for is returned.

(6) Remaining time until timeout (lefttmo) *

If tskstat is TTS_WAI or TTS_WAS and if tskwait is other than TTW_DLY, the remaining wait time of the target task is returned. If the time-out will be occurred at the next time-tick, 0 is returned.

For an endless wait, TMO_FEVR is returned.

For TTW_DLY (WAITING state by dly_tsk), the lefttmo is indeterminate.

(7) Activation request count (actcnt)

The number of currently queued activation requests is returned.

(8) Wakeup request count (wupcnt) *

The number of currently queued wakeup requests is returned.

(9) Suspension count (suscnt) *

The number of currently nested suspension requests is returned.

5.5.12 Refers to Task State (Simplified Version) (ref_tst, iref_tst)

❑C Language API

```
ER        ref_tst(ID tskid, T_RTST *pk_rtst);
ER        iref_tst(ID tskid, T_RTST *pk_rtst);
```

❑Parameter

```
tskid     Task ID
pk_rtst   Pointer to the storage to which the task state is returned
```

❑Packet Structure

```
typedef   struct t_rtst {
          STAT    tskstat;    Task state
          STAT    tskwait;    Reason for waiting
} T_RTST;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

```
E_PAR              Parameter error
                   pk_rtst == NULL
E_ID               Invalid ID number
                   (1) tskid<0, VTMAX_TSK < tskid
                   (2) In an invocations from non-task context, tskid == 0
E_CTX              Context error (invoked from system state not permitted)
                     Note : The E_CTX is not detected in the following cases.
                         (1) Invocation of ref_tst from non-task context
                         (2) Invocation of iref_tst from task context
E_MACV             Memory access violation (only for ref_tst)
                   The operand write access to the area indicated by pk_rtst has not been
                   permitted to the invoking task.
E_NOEXS            Non-existent object
                   The task indicated by tskid does not exist.
```

❑Function

This service call is simplified version of ref_tsk and iref_tsk. The same value returned by ref_tsk and iref_tsk rhrough tskstat and tskwait apply to ref_tst and iref_tst as well.

In ref_tst, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

5.6 Task Dependent Synchronization Function

Table 5.6 shows specifications of the task dependent synchronization function.

Table 5.6 Specifications of the Task Synchronization Function

No.	Item	Content
1	Maximum number of queued task wakeup request	255
2	Maximum number of nested task suspension request	1

Table 5.7 Service Calls for Task Dependent Synchronization

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	slp_tsk	[S][B]	Puts task to sleep	T		E		U	
2	tslp_tsk	[S]	Puts task to sleep (with Timeout)	T		E		U	
3	wup_tsk	[S][B]	Wakes up task	T		E	D	U	
4	iwup_tsk	[S][B]			D	E	D	U	
5	can_wup	[S][B]	Cancels task wakeup request	T		E	D	U	
6	ican_wup				D	E	D	U	
7	rel_wai	[S][B]	Release task from WAITING state	T		E	D	U	
8	irel_wai	[S][B]			D	E	D	U	
9	sus_tsk	[S][B]	Suspend task	T		E	D	U	
10	isus_tsk				D	E	D	U	
11	rsm_tsk	[S][B]	Resume suspended task	T		E	D	U	
12	irsm_tsk				D	E	D	U	
13	frsm_tsk	[S]	Forcibly resume suspended task	T		E	D	U	
14	ifrsm_tsk				D	E	D	U	
15	dly_tsk	[S][B]	Delay task	T		E		U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings :

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.6.1 Puts Task to Sleep (slp_tsk, tslp_tsk)

❑C Language API

```
ER          slp_tsk(void);
ER          tslp_tsk(TMO tmout);
```

❑Parameter

```
<Only for tslp_tsk>
tmout       Timeout (millisecond)
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error (only for tslp_tsk) tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_RLWAI	Forced release from the waiting Accept rel_wai or irel_wai while waiting
E_TMOUT	Polling failure or timeout (only for tslp_tsk)

❑Function

Places the invoking task into a wakeup wait state. However, if any wakeup request for the invoking task is queued, the number of queued wakeup requests is decremented by one and execution is continued as it is.

The task is released from the wakeup wait state by the wup_tsk or iwup_tsk service call. In this case, this service call is terminated normally.

For the tslp_tsk service call, specify a wait time in tmout.

If a positive value is specified for tmout and the specified tmout time elapses while in the WAITING state, the task is placed out of the wait state and E_TMOUT is returned as error code.

If, when tmout = TMO_POL (= 0) is specified, the number of queued wakeup requests is greater than 0, the number of queued wakeup requests is decremented by 1 and execution is continued; if the number of queued requests is 0, E_TMOUT is returned as error code.

If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. In this case, the tslp_tsk service call operates the same way as slp_tsk.

5.6.2 Wakeup Task (wup_tsk, iwup_tsk)

❑C Language API

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

❑Parameter

tskid	Task ID

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number (1) tskid<0, VTMAX_TSK < tskid (2) In an invocations from non-task context, tskid == 0
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for wup_tsk) Stack pointer points out of user stack for invoking task.
E_OBJ	Object state error The task indicated by tskid is in the DORMANT state.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.
E_QOVR	Queuing overflow Wakeup request count has already reached the maximum value.

❑Function

Releases the WAITING state by an invocation of the slp_tsk or tslp_tsk service call.

In wup_tsk, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

When the task is not in the WAITING state by an invocation of the slp_tsk or tslp_tsk service call, up to 255 task wakeup requests from service calls wup_tsk and iwup_tsk can be queued.

5.6.3 Cancels Task Wakeup Request (can_wup, ican_wup)

❑C Language API

```
ER_UINT    can_wup(ID tskid);
ER_UINT    ican_wup(ID tskid);
```

❑Parameter

```
tskid      Task ID
```

❑Return Value

```
Wakeup request count (positive value or 0), or error code
```

❑Error Code

```
E_ID              Invalid ID number
                  (1) tskid<0, VTMAX_TSK < tskid
                  (2) In an invocations from non-task context, tskid == 0
E_CTX             Context error (invoked from system state not permitted)
                    Note : The E_CTX is not detected in the following cases.
                        (1) Invocation of can_wup from non-task context
                        (2) Invocation of ican_wup from task context
E_OBJ             Object state error
                  The task indicated by tskid is in the DORMANT state.
E_NOEXS           Non-existent object
                  The task indicated by tskid does not exist.
```

❑Function

Gets a count of wakeup requests that have been queued in the task indicated by tskid and returns the result as return parameter, at the same time invalidating all of those wakeup requests.

In can_wup, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

5.6.4 Forcibly Releases Task from WAITING State (rel_wai, irel_wai)

❑C Language API

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

❑Parameter

```
tskid     Task ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

E_ID	Invalid ID number tskid<=0, VTMAX_TSK < tskid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for rel_wai) Stack pointer points out of user stack for invoking task.
E_OBJ	Object state error The task indicated by tskid is in the DORMANT state.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.

❑Function

If the task indicated by tskid is in some kind of WAITING state (except SUSPENDED state), this service call forcibly releases the WAITING state. The task freed by this service call has E_RLWAI returned to it as error code.

If this service call is invoked for a task that is in the WAITING-SUSPENDED state, the target task goes to the SUSPENDED state. Then, when the rsm_tsk or irsm_tsk, or the frsm_tsk or ifrsm_tsk service call is invoked, the target task is released from the SUSPENDED state, in which case the task has E_RLWAI returned to it as error code.

When the task at the top of a message buffer send queue or a variable-sized memory pool memory acquisition queue is forcibly dequeued, it is possible that other tasks (waiting for message buffer transmission or variable-sized memory pool memory acquisition) will be released from the wait state.

To release tasks from the SUSPENDED state, use rsm_tsk, irsm_tsk, frsm_tsk, or ifrsm_tsk.

5.6.5 Suspends Task (sus_tsk, isus_tsk)

❑C Language API

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

❑Parameter

tskid	Task ID

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number (1) tskid<0, VTMAX_TSK < tskid (2) In an invocations from non-task context, tskid == 0
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for sus_tsk) Stack pointer points out of user stack for invoking task.
E_OBJ	Object state error (1)The task indicated by tskid is in the DORMANT state. (2)When issuing isus_tsk in the dispatching-disabled state, tskid indicates the task in the RUNING state.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.
E_QOVR	Queuing overflow Suspension count has already reached the maximum value.

❑Function

Suspends execution of the task indicated by tskid and places it into the SUSPENDED state. If the task indicated by tskid is in the WAITING state when this service call is invoked, it goes to the WAITING-SUSPENDED state. At this time, the number of nested suspension requests changes from 0 to 1. If the target task is already in the SUSPENDED state or WAITING-SUSPENDED state, error E_QOVR is returned. At this time, the number of nested suspension requests remains unchanged (= 1). This is because the number of nested suspension requests by this service call is 1 at the most.

In sus_tsk, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified. However, if, while in a dispatching-disabled state, TSK_SELF or the invoking task ID is specified for tskid when the sus_tsk service call is invoked, error E_CTX is returned.

Tasks are released from the SUSPENDED state by an invocation of the rsm_tsk, irsm_tsk, frsm_tsk or ifrsm_tsk service call.

5.6.6 Resumes Suspended Task (rsm_tsk, irsm_tsk), Forcibly Resumes Suspended Task (frsm_tsk, ifrsm_tsk)

❑C Language API

```
ER        rsm_tsk(ID tskid);
ER        irsm_tsk(ID tskid);
ER        frsm_tsk(ID tskid);
ER        ifrsm_tsk(ID tskid);
```

❑Parameter

```
tskid     Task ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID             Invalid ID number
                 (1) tskid<0, VTMAX_TSK < tskid
                 (2) In an invocations from non-task context, tskid == 0
E_CTX            Context error (invoked from system state not permitted)
E_MACV           Memory access violation (only for rsm_tsk and frsm_tsk)
                 Stack pointer points out of user stack for invoking task.
E_OBJ            Object state error
                 The task indicated by tskid is in the DORMANT state.
E_NOEXS          Non-existent object
                 The task indicated by tskid does not exist.
```

❑Function

Releases the task indicated by tskid from the SUSPENDED state.

More specifically, the rsm_tsk and irsm_tsk service calls operate in such a way that if the task indicated by tskid is in the SUSPENDED state when either call is invoked, the number of nested suspension requests is decremented by 1. Since the number of nested suspension requests is 1 at the most, the number of nested suspension requests is thereby decremented to 0, resulting in the SUSPENDED being removed.

The frsm_tsk and ifrsm_tsk service calls decrement the number of nested suspension requests to 0. In this kernel, since the number of nested suspension requests is 1 at the most, these service calls behave the same way as rsm_tsk and irsm_tsk.

5.6.7 Delays Task (dly_tsk)

❑C Language API

```
ER        dly_tsk(RELTIM dlytim);
```

❑Parameter

```
dlytim    Delayed time (in milliseconds)
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

E_PAR	Parameter error (1) (0x7FFFFFFF - TIC_NUME)/TIC_DENO < dlytim
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_RLWAI	Forced release from the waiting Accept rel_wai or irel_wai while waiting

❑Function

Changes the status of the invoking task from an execution state to a lapse of time wait state, waiting for the time specified by dlytim to elapse. When the time specified by dlytim has elapsed, the status of the invoking task is changed back to the READY state. If dlytim = 0 is specified and in this case too, the invoking task is placed into the WAITING state.

This service call, unlike the tslp_tsk service call, is terminated normally when it has finished off by delaying execution by an amount of time, dlytim. Note also that even when the wup_tsk or iwup_tsk service call is executed during the delay time, the wait state is not exited. It is only when the rel_wai, irel_wai or the ter_tsk service call is invoked that the wait state is exited before the delay time elapses.

5.7 Task Exception Handling Function

Table 5.8 shows specifications of the task exception handling function.

Table 5.8 Specifications of the Task Exception Handling Function

No.	Item	Content
1	Exception code	32 bits
2	Start Condition of task exception handling routine	When all the following conditions are satisfied; 1. Corresponded task is in task exception enabled state. 2. Pending exception code is not 0. 3. Non-task context is not executed. 4. Corresponded task is in RUNNING state.

Table 5.9 Service Calls for Task Exception Handling Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	def_tex		Defines task exception handling routine	T		E	D	U	
2	ras_tex	[S]	Raises task exception handling	T		E	D	U	
3	iras_tex	[S]			N	E	D	U	
4	dis_tex	[S]	Disables task exception	T		E	D	U	
5	ena_tex	[S]	Enables task exception	T		E	D	U	
6	sns_tex	[S]	Refers to task exception disabled state	T	N	E	D	U	L
7	ref_tex		Refers to task exception handling state	T		E	D	U	
8	iref_tex				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings :
"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.7.1 Defines Task Exception Handling Routine (def_tex)

❑C Language API

```
ER        def_tex(ID tskid, T_DTEX *pk_dtex);
```

❑Parameter

tskid	Task ID
pk_ctex	Pointer to the packet containing the task exception handling routine definition information

❑Packet Structure

```
typedef   struct t_dtex {
          ATR     texatr;          Task exception handling routine attribute
          FP      texrtn;          Task exception handling routine start address
} T_DTEX;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_RSATR	Reserved attribute texatr != TA_HLNG.
E_PAR	Parameter error texrtn == NULL
E_ID	Invalid ID number tskid <0, VTMAX_TSK < tskid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_dtex has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

The def_tex defines a task exception handling routine for the task indicated by tskid according to the content of pk_dtex.

If a task exception handling routine has already been defined, the previous definition is cancelled and is replaced with the new definition.

If pk_dtex = NULL(=0) is specified, the definition of the task exception handling routine for tskid is cancelled. At this time the task pending exception code is cleared to 0, and the task exception handling is disabled.

(1) Task ID (tskid)

Specify task ID to define a task exception handling routine.

Specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

(2) Task Exception Handling Routine Attribute (texatr)

Only TA_HLNG can be specified for texatr.

- TA_HLNG (0x0000)
  Only C-language is supported for task exception handling routine description language.

(3) Task Exception Handling Routine Start Address (texrtn)

Specify the task exception handling routine start address.

5.7.2 Raises Task Exception Handling (ras_tex, iras_tex)

❑C Language API

```
ER        ras_tex(ID tskid, TEXPTN rasptn);
ER        iras_tex(ID tskid, TEXPTN rasptn);
```

❑Parameter

tskid	Task ID
rasptn	Task exception code to be requested

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error rasptn == 0
E_ID	Invalid ID number (1) tskid<0, VTMAX_TSK < tskid (2) In an invocations from non-task context, tskid == 0
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for ras_tex) Stack pointer points out of user stack for invoking task.
E_OBJ	Object state error (1) The task indicated by tskid is in the DORMANT state. (2) A task exception handling routine is not defined for the task indicated by tskid.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.

❑Function

This service call requests task exception handling for the task indicated by tskid. The task pending exception code for the task is ORed with the value indicated by rasptn.

In ras_tex, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

When the conditions for starting task exception handling routine are satisfied, the task exception handling routine is initiated. Please refer to "3.5 Task Exception Handling" for the conditions for starting task exception handling routine.

When a task exception handling routine is initiated, the task pending exception code is cleared to 0, and the task exception handling is disabled. The pending exception code before clear and extended information for the task are passed to the task exception handling routine.

At the return from a task exception handling routine, the task exception is enabled, and the task restarts execution from the point immediately before the start of the task exception handling routine.

When a task exception handling routine is initiated, the kernel saves context registers for the task to the user stack. If the user stack is overflow, system goes down.

It is necessary to release from CPU locked state by the end of a task exception handling routine when shifting to the CPU locked state in a task exception handling routine. If CPU is locked at the end of a task exception handling routine, system goes down.

The interrupt priority level (PSW.IPL) before and after the start of a task exception handling routine is not changed. And the interrupt priority level before and after the return from a task exception handling routine is not changed. When the interrupt priority level at the end of a task exception handling routine is higher than the kernel interrupt mask level, system goes down.

5.7.3 Disables Task Exception (dis_tex)

❑C Language API

```
ER        dis_tex(void);
```

❑Parameter

None

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)
E_OBJ	Object state error A task exception handling routine is not defined for the invoking task.

❑Function

This service call disables task exception for the invoking task.

5.7.4 Enables Task Exception (ena_tex)

❑C Language API

```
ER          ena_tex(void);
```

❑Parameter

None

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OBJ	Object state error A task exception handling routine is not defined for the invoking task.

❑Function

This service call enables task exception for the invoking task.

5.7.5 Refers to Task Exception Disabled State (sns_tex)

❑C Language API

```
BOOL        sns_tex(void);
```

❑Parameter

None

❑Return Value

TRUE or FALSE or error code

❑Error Code

```
E_CTX            Context error (invoked from system state not permitted)
```

❑Function

This service call returns FALSE if the task in the RUNNING state is in the task exception enabled state, and otherwise returns TRUE.

Table 5.10 Return Value of sns_tex

The task in the RUNNING state	Task exception handling routine for the task in the RUNNING state	Task exception disabled state for the task in the RUNNING state	Return value	Note
Exist	Defined	Enabled state	FALSE	
		Disabled state	TRUE	
	Not defined	Disabled state	TRUE	The task exception is disabled when a task exception handling routine is not defined.
Not exist	-	-	TRUE	

5.7.6 Refers to Task Exception Handling State (ref_tex, iref_tex)

❑C Language API

```
ER        ref_tex(ID tskid, T_RTEX *pk_rtex);
ER        iref_tex(ID tskid, T_RTEX *pk_rtex);
```

❑Parameter

tskid	Task ID
pk_rtex	Pointer to the storage to which the task exception handling state is returned

❑Packet Structure

```
typedef    struct t_rtex {
           STAT    texstat;     Task exception handling state
           TEXPTN  texptn;      Pending exception code
} T_RTEX;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error pk_rtex == NULL
E_ID	Invalid ID number (1) tskid<0, VTMAX_TSK < tskid (2) In an invocations from non-task context, tskid == 0
E_CTX	Context error (invoked from system state not permitted) Note : The E_CTX is not detected in the following cases. (1) Invocation of ref_tex from non-task context (2) Invocation of iref_tex from task context
E_MACV	Memory access violation (only for ref_tex) The operand write access to the area indicated by pk_rtex has not been permitted to the invoking task.
E_OBJ	Object state error (1) The task indicated by tskid is in the DORMANT state. (2) A task exception handling routine is not defined for the task indicated by tskid.
E_NOEXS	Non-existent object The task indicated by tskid does not exist.

❑Function

Refers to the state of the task exception handling indicated by tskid and returns it to the area pointed to by pk_rtex.

In ref_tex, specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

The area pointed to by pk_rtex has one of the following values returned to it.

(1) Task exception handling state (tskstat)

Either of the following value is returned.

- TTEX_ENA (= 0) : Task exception enabled state
- TTEX_DIS (= 1) : Task exception disabled state

(2) Pending exception code (texptn)

Pending exception code is returned.

5.8 Synchronization and Communication Function (Semaphore)

Table 5.11 shows specifications of the semaphore function.

Table 5.11 Specifications of the Semaphore Function

No.	Item	Content
1	Semaphore ID	1 - VTMAX_SEM *1
2	Maximum semaphore count	1 - TMAX_MAXSEM *2
3	Semaphore attributes	TA_TFIFO : Task wait queue is managed in FIFO order. TA_TPRI : Task wait queue is managed in task current priority order.

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates maximum semaphore ID.

2 This is the macro defined in kernel.h. The definition is 65535.

Table 5.12 Service Calls for Semaphore Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_sem		Creates semaphore	T		E	D	U	
2	acre_sem		Creates semaphore (Automatic ID Assignment)	T		E	D	U	
3	del_sem		Deletes semaphore	T		E	D	U	
4	sig_sem	[S][B]	Releases semaphore resource	T		E	D	U	
5	isig_sem	[S][B]			N	E	D	U	
6	wai_sem	[S][B]	Acquire semaphore resource	T		E		U	
7	pol_sem	[S][B]	Acquire semaphore resource (polling)	T		E	D	U	
8	ipol_sem				N	E	D	U	
9	twai_sem	[S]	Acquire semaphore resource (with timeout)	T		E		U	
10	ref_sem		Refers to semaphore state	T		E	D	U	
11	iref_sem				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings :

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.8.1 Creates Semaphore (cre_sem, acre_sem)

❑C Language API

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

❑Parameter

semid	Semaphore ID
pk_csem	Pointer to the packet containing the semaphore creation information

❑Packet Structure

```
typedef    struct t_csem {
           ATR     sematr;           Semaphore attribute
           UINT    isemcnt;          Initial semaphore count
           UINT    maxsem;           Maximum semaphore count
} T_CSEM;
```

❑Return Value

In cre_sem : E_OK for normal completion or error code
In acre_sem : Created semaphore ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute Either of bits except bit0 of sematr is 1.
E_PAR	Parameter error (1) pk_csem == NULL (2) maxsem <= 0, TMAX_MAXSEM < maxsem (3) maxsem < isemcnt
E_ID	Invalid ID number (only for cre_sem) semid<=0, VTMAX_SEM < semid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_csem has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOID	No ID number available (only for acre_sem)
E_OBJ	Object state error (only for cre_sem) The semaphore indicated by semid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_sem creates a semaphore with semaphore ID indicated by semid according to the content of pk_csem. The acre_sem creates a semaphore according to the content of pk_csem, and returns the created semaphore ID.

(1) Semaphore ID (semid)

The cre_sem creates a semaphore with the semaphore ID indicated by semid.

(2) Semaphore Attribute (sematr)

The following are specified for sematr.

```
sematr:= ( TA_TFIFO || TA_TPRI )
```

- TA_TFIFO (0x0000)
  Task wait queue is managed in FIFO order.
- TA_TPRI (0x0001)
  Task wait queue is managed in task current priority order.

(3) Initial Semaphore Count (isemcnt)

Specify initial semaphore count within the range from 0 to maxsem.

(4) Maximum Semaphore Count (maxsem)

Specify maximum semaphore count.

5.8.2 Deletes Semaphore (del_sem)

❑C Language API

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

❑Parameter

semid	Semaphore ID

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number semid<=0, VTMAX_SEM < semid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The semaphore indicated by semid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the semaphore indicated by semid.

No error will occur even if there is waiting task for the semaphore indicated by semid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

5.8.3 Releases Semaphore Resource (sig_sem, isig_sem)

❑C Language API

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

❑Parameter

```
semid     Semaphore ID
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number semid<=0, VTMAX_SEM < semid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for sig_sem) Stack pointer points out of user stack for invoking task.
E_NOEXS	Non-existent object The semaphore indicated by semid does not exist.
E_QOVR	Queuing overflow Semaphore count has already reached the maximum count.

❑Function

Returns one resource to the semaphore indicated by semid. If the target semaphore has any task waiting for it, the task at the top of the semaphore queue is assigned the resource, upon which the task is dequeued. If there are no tasks waiting for the semaphore, the count value of the semaphore is incremented by 1.

Note that the maximum value of the semaphore count is defined at creation of the semaphore.

5.8.4 Acquires Semaphore Resource (wai_sem, pol_sem, ipol_sem, twai_sem)

❑C Language API

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

❑Parameter

```
semid     Semaphore ID
<Only for twai_sem>
tmout     Timeout (millisecond)
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	Invalid ID number semid<=0, VTMAX_SEM < semid
E_CTX	Context error (invoked from system state not permitted) Note : The E_CTX is not detected in the following cases. (1) Invocation of pol_sem from non-task context (2) Invocation of ipol_sem from task context
E_MACV	Memory access violation (only for wai_sem and twai_sem) Stack pointer points out of user stack for invoking task.
E_NOEXS	Non-existent object The semaphore indicated by semid does not exist.
E_RLWAI	Forced release from the waiting (only for wai_sem and twai_sem) Accept rel_wai or irel_wai while waiting
E_TMOUT	Polling failure or timeout
E_DLT	Waiting object deleted (only for wai_sem and twai_sem) The semaphore indicated by semid has been deleted while waiting.

❑Function

Acquires one resource from the semaphore indicated by semid.

If the target semaphore has 1 or more resources, the number of resources of the semaphore is decremented by 1 and execution is continued. If the number of resource is 0 and the service call concerned is wai_sem or twai_sem, then the invoking task is tied to the semaphore queue; or, in the case of the pol_sem or ipol_sem service call, it immediately returns error E_TMOUT. The queue is managed according to the attributes specified when the semaphore was created.

For the twai_sem service call, specify a wait time in tmout. If a positive value is specified for tmout and the specified tmout time elapses while the wait release condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with pol_sem. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. In this case, the service call operates the same way as with wai_sem.

5.8.5 Refers to Semaphore State (ref_sem, iref_sem)

❑C Language API

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

❑Parameter

```
semid     Semaphore ID
pk_rsem   Pointer to the storage to which the semaphore state is returned
```

❑Packet Structure

```
typedef   struct t_rsem {
          ID      wtskid;     The task ID at the top of the task wait queue
          UINT    semcnt;     Current semaphore count
} T_RSEM;
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR             Parameter error
                  pk_rsem == NULL
E_ID              Invalid ID number
                  semid<=0, VTMAX_SEM < semid
E_CTX             Context error (invoked from system state not permitted)
                    Note : The E_CTX is not detected in the following cases.
                        (1) Invocation of ref_sem from non-task context
                        (2) Invocation of iref_sem from task context
E_MACV            Memory access violation (only for ref_sem)
                  The operand write access to the area indicated by pk_rsem has not been
                  permitted to the invoking task.
E_NOEXS           Non-existent object
                  The semaphore indicated by semid does not exist.
```

❑Function

Refers to the state of the semaphore indicated by semid.

The area pointed to by pk_rsem will have returned to it the task ID at the top of the queue (wtskid) and the current semaphore count value (semcnt).

If the target semaphore has no tasks waiting for it, TSK_NONE (= 0) is returned as the waiting task ID.

5.9 Synchronization and Communication Function (Eventflag)

Table 5.13 shows specifications of the eventflag function.

Table 5.13 Specifications of the Eventflag Function

No.	Item	Content
1	Eventflag ID	1 - VTMAX_FLG *1
2	Eventflag size	32 bits
3	Eventflag attribute	TA_TFIFO : Task wait queue is managed in FIFO order. TA_TPRI : Task wait queue is managed in task current priority order. *2 TA_WSGL : Does not permit multiple tasks to wait for the eventflag. TA_WMUL : Permits multiple tasks to wait for the eventflag. TA_CLR : Clears the eventflag at the time of waiting release.

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates maximum eventflag ID.

2 If TA_CLR attribute is not specified, even if there is the TA_TPRI attribute specified, the queue is managed in the same way as for the TA_TFIFO attribute. This behavior falls outside µITRON 4.0 specification.

Table 5.14 Service Calls for Eventflag Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_flg		Creates eventflag	T		E	D	U	
2	acre_flg		Creates eventflag (Automatic ID Assignment)	T		E	D	U	
3	del_flg		Deletes eventflag	T		E	D	U	
4	set_flg	[S][B]	Sets eventflag	T		E	D	U	
5	iset_flg	[S][B]			N	E	D	U	
6	clr_flg	[S][B]	Clears eventflag	T		E	D	U	
7	iclr_flg				N	E	D	U	
8	wai_flg	[S][B]	Waits for eventflag	T		E		U	
9	pol_flg	[S][B]	Waits for eventflag (polling)	T		E	D	U	
10	ipol_flg				N	E	D	U	
11	twai_flg	[S]	Waits for eventflag (with timeout)	T		E		U	
12	ref_flg		Refers to eventflag state	T		E	D	U	
13	iref_flg				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings :

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.9.1 Creates Eventflag (cre_flg, acre_flg)

❑C Language API

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

❑Parameter

flgid	Eventflag ID
pk_cflg	Pointer to the packet containing the eventflag creation information

❑Packet Structure

```
typedef   struct t_cflg {
          ATR     flgatr;           Eventflag attribute
          FLGPTN  iflgptn;          Initial value of event flag
} T_CFLG;
```

❑Return Value

In cre_flg : E_OK for normal completion or error code

In acre_flg : Created eventflag ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute Either of bits except bit0, bit1 and bit2 of flgatr is 1.
E_PAR	Parameter error pk_cflg == NULL
E_ID	Invalid ID number (only for cre_flg) flgid<=0, VTMAX_FLG < flgid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_cflg has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOID	No ID number available (only for acre_flg)
E_OBJ	Object state error (only for cre_flg) The eventflag indicated by flgid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_flg creates a eventflag with eventflag ID indicated by flgid according to the content of pk_cflg. The acre_flg creates a eventflag according to the content of pk_cflg, and returns the created eventflag ID.

(1) Eventflag ID (flgid)

The cre_flg creates a eventflag with the eventflag ID indicated by flgid.

(2) Eventflag Attribute (flgatr)

The following are specified for flgatr.

```
flgatr:= ( ( TA_TFIFO || TA_TPRI ) | ( TA_WSGL || TA_WMUL ) | [TA_CLR] )
```

- TA_TFIFO (0x0000)
  Task wait queue is managed in FIFO order.
- TA_TPRI (0x0001)
  Task wait queue is managed in task current priority order.
  When TA_CLR attribute is not specified, even if there is the TA_TPRI attribute specified, the queue is managed in the same way as for the TA_TFIFO attribute. This behavior falls outside µITRON 4.0 specification.
- TA_WSGL (0x0000)
  Does not permit multiple tasks to wait for the eventflag.
- TA_WMUL(0x0002)
  Permits multiple tasks to wait for the eventflag.
- TA_CLR(0x0004)
  Clears the eventflag at the time of waiting release.

(3) Initial value of event flag (iflgptn)

Specify initial value of the eventflag.

5.9.2 Deletes Eventflag (del_flg)

❑C Language API

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

❑Parameter

```
flgid     Eventflag ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID              Invalid ID number
                  flgid<=0, VTMAX_FLG < flgid
E_CTX             Context error (invoked from system state not permitted)
E_MACV            Memory access violation
                  Stack pointer points out of user stack for invoking task.
E_OACV            Object access violation
                  The invoking task does not belong to trusted domain.
E_NOEXS           Non-existent object
                  The eventflag indicated by flgid does not exist.
```

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the eventflag indicated by flgid.

No error will occur even if there is waiting task for the eventflag indicated by flgid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

5.9.3 Sets Eventflag (set_flg, iset_flg)

❑C Language API

```
ER        set_flg(ID flgid, FLGPTN setptn);
ER        iset_flg(ID flgid, FLGPTN setptn);
```

❑Parameter

flgid	Eventflag ID
setptn	Bit pattern to set

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number flgid<=0, VTMAX_FLG < flgid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for set_flg) Stack pointer points out of user stack for invoking task.
E_NOEXS	Non-existent object The eventflag indicated by flgid does not exist.

❑Function

These service calls updates the eventflag indicated by flgid to the one logically OR'ed with the value indicated by setptn.

After that, these service calls evaluate whether the wait condition of the tasks in the wait queue is satisfied. This evaluation is done in order of the wait queue. If the wait condition is satisfied, the relevant task is released from WAITING state. At this time, the bit pattern of the target event flag is cleared to 0 and this service call finishes processing if the TA_CLR attribute is specified for the target eventflag.

If the eventflag has the TA_WMUL attribute specified but does not have the TA_CLR attribute specified, it is possible that multiple tasks will be released from the WAITING states by one invocation of set_flg or iset_flg. If there are multiple tasks to be released from the WAITING states, they are freed in the order they are tied up in the task wait queue.

5.9.4 Clears Eventflag (clr_flg, iclr_flg)

❑C Language API

```
ER        clr_flg(ID flgid, FLGPTN clrptn);
ER        iclr_flg(ID flgid, FLGPTN clrptn);
```

❑Parameter

```
flgid     Eventflag ID
clrptn    Bit pattern to clear
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID              Invalid ID number
                  flgid<=0, VTMAX_FLG < flgid
E_CTX             Context error (invoked from system state not permitted)
                    Note : The E_CTX is not detected in the following cases.
                        (1) Invocation of clr_flg from non-task context
                        (2) Invocation of iclr_flg from task context
E_NOEXS           Non-existent object
                  The eventflag indicated by flgid does not exist.
```

❑Function

Updates the eventflag indicated by flgid to the one logically AND'ed with the value indicated by clrptn.

If the bits of clrptn are all 1's, no operation will be performed on the eventflag, but no errors are assumed.

5.9.5 Waits for Eventflag (wai_flg, pol_flg, ipol_flg, twai_flg)

❑C Language API

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

❑Parameter

flgid	Eventflag ID
waiptn	Wait bit pattern
wfmode	Wait mode
p_flgptn	Pointer to the storage to which the bit pattern at waiting release is returned
<Only for twai_flg>	
tmout	Timeout (millisecond)

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error (1) p_flgptn == NULL (2) waiptn=0 (3) wfmode is invalid (4) tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	Invalid ID number flgid<=0, VTMAX_FLG < flgid
E_CTX	Context error (invoked from system state not permitted) Note : The E_CTX is not detected in the following cases. (1) Invocation of pol_flg from non-task context (2) Invocation of ipol_flg from task context
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task.(only for wai_flg, and twai_flg) (2) The operand write access to the area indicated by p_flgptn has not been permitted to the invoking task. (only for wai_flg, pol_flg and twai_flg)
E_ILUSE	Illegal use of service call There is a waiting task for the eventflag with TA_WSGL attribute indicated by flgid.
E_NOEXS	Non-existent object The eventflag indicated by flgid does not exist.
E_RLWAI	Forced release from the waiting (only for wai_flg and twai_flg) Accept rel_wai or irel_wai while waiting
E_TMOUT	Polling failure or timeout
E_DLT	Waiting object deleted (only for wai_flg and twai_flg) The eventflag indicated by flgid has been deleted while waiting.

❑Function

Waits for the bit pattern of the eventflag indicated by flgid to satisfy the wait release condition specified by waiptn and wfmode. The area pointed to by p_flgptn will have returned to it the eventflag bit pattern that satisfied the above condition.

If, when this service call is invoked, the wait release condition is already met, the service call is immediately completed. If the wait release condition is not met yet and the service call concerned is wai_flg or twai_flg, the task is tied to the task wait queue; or, in the case of the pol_flg and ipol_flg service calls, it immediately returns error E_TMOUT.

If TA_TFIFO attribute is specified when created, the queue is managed in FIFO order.

On the other hand, if TA_TPRI attribute is specified, the queue is managed in task current priority order. Among tasks with the same priority, the queue is managed in FIFO order.

However, if TA_CLR attribute is not specified, even if there is the TA_TPRI attribute specified, the queue is managed in the same way as for the TA_TFIFO attribute. This behavior falls outside μITRON 4.0 specification.

Note that if TA_WSGL attribute is specified, because in no case can multiple tasks wait for an eventflag at the same time, no differences exist between TA_TFIFO and TA_TPRI.

For wfmode, specify as follows :

```
wfmode  : = ( (TWF_ANDW||TWF_ORW) )
```

- TWF_ANDW (0x00000000) : AND wait
- TWF_ORW (0x00000001) : OR wait

For TWF_ANDW, the service call waits for all of the bits specified by waiptn to be set. For TWF_ORW, the service call waits for one of the bits specified by waiptn in the eventflag indicated by flgid to be set.

For the twai_flg service call, specify a wait time in tmout. If a positive value is specified for tmout and the specified tmout time elapses while the wait condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with pol_flg. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. In this case, the service call operates the same way as with wai_flg.

5.9.6 Refers to Eventflag State (ref_flg, iref_flg)

❑C Language API

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

❑Parameter

```
flgid     Eventflag ID
pk_rflg   Pointer to the storage to which the eventflag state is returned
```

❑Packet Structure

```
typedef   struct t_rflg {
          ID      wtskid;     The task ID at the top of the task wait queue
          FLGPTN  flgptn;     Evetnflag's current bit pattern
} T_RFLG;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

```
E_PAR              Parameter error
                   pk_rflg == NULL
E_ID               Invalid ID number
                   flgid<=0, VTMAX_FLG < flgid
E_CTX              Context error (invoked from system state not permitted)
                     Note : The E_CTX is not detected in the following cases.
                         (1) Invocation of ref_flg from non-task context
                         (2) Invocation of iref_flg from task context
E_MACV             Memory access violation (only for ref_flg)
                   The operand write access to the area indicated by pk_rflg has not been
                   permitted to the invoking task.
E_NOEXS            Non-existent object
                   The eventflag indicated by flgid does not exist.
```

❑Function

Refers to the state of the eventflag indicated by flgid.

The area pointed to by pk_rflg will have the task ID at the top of the queue (wtskid) and the current bit pattern of the eventflag (flgptn) returned to it.

If there are no tasks waiting for the target eventflag, TSK_NONE (= 0) is returned as the waiting task ID.

5.10 Synchronization and Communication Function (Data Queue)

Table 5.15 shows specifications of the data queue function.

Table 5.15 Specifications of the Data Queue Function

No.	Item	Content
1	Data queue ID	1 - VTMAX_DTQ *1
2	Data size	4 bytes
3	Number of data elements)	Up to 65535
4	Data queue attribute	TA_TFIFO : Task wait queue for sending is managed in FIFO order. TA_TPRI : Task wait queue for sending is managed in task current priority order.

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates the maximum data queue ID.

Table 5.16 Service Calls for Data Queue Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_dtq		Creates data queue	T		E	D	U	
2	acre_dtq		Creates data queue (Automatic ID Assignment)	T		E	D	U	
3	del_dtq		Deletes data queue	T		E	D	U	
4	snd_dtq	[S]	Sends to data queue	T		E		U	
5	psnd_dtq	[S]	Sends to data queue (polling)	T		E	D	U	
6	ipsnd_dtq	[S]			N	E	D	U	
7	tsnd_dtq	[S]	Sends to data queue (with timeout)	T		E		U	
8	fsnd_dtq	[S]	Forced sends to data queue	T		E	D	U	
9	ifsnd_dtq	[S]			N	E	D	U	
10	rcv_dtq	[S]	Receives from data queue	T		E		U	
11	prcv_dtq	[S]	Receives from data queue (polling)	T		E	D	U	
12	iprcv_dtq				N	E	D	U	
13	trcv_dtq	[S]	Receives from data queue (with timeout)	T		E		U	
14	ref_dtq		Refers to data queue state	T		E	D	U	
15	iref_dtq				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings :
"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.10.1 Creates Data Queue (cre_dtq, acre_dtq)

❑C Language API

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

❑Parameter

dtqid	Data queue ID
pk_cdtq	Pointer to the packet containing the data queue creation information

❑Packet Structure

```
typedef   struct t_cdtq {
          ATR     dtqatr;           Data queue attribute
          UINT    dtqcnt;           Capacity of the data queue area (the number of data elements)
          UINT    dtq;              Start address of the data queue area
} T_CDTQ;
```

❑Return Value

In cre_dtq : E_OK for normal completion or error code
In acre_dtq : Created data queue ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute Either of bits except bit0 is 1.
E_PAR	Parameter error (1) pk_cdtq == NULL (2) dtqcnt > 65535 (3) dtqcnt != 0 and dtq + TSZ_DTQ(dtqcnt) > 0x100000000
E_ID	Invalid ID number (only for cre_dtq) dtqid<=0, VTMAX_DTQ < dtqid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_cdtq has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOMEM	Insufficient memory dtqcnt!=0 and dtq == NULL
E_NOID	No ID number available (only for acre_dtq)
E_OBJ	Object state error (only for cre_dtq) The data queue indicated by dtqid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_dtq creates a data queue with data queue ID indicated by dtqid according to the content of pk_cdtq. The acre_dtq creates a data queue according to the content of pk_cdtq, and returns the created data queue ID.

(1) Data Queue ID (dtqid)

The cre_dtq creates a data queue with the data queue ID indicated by dtqid.

(2) Data Queue Attribute (dtqatr)

The following are specified for dtqatr.

```
dtqatr:= ( TA_TFIFO || TA_TPRI )
```

- TA_TFIFO (0x0000)
  Task wait queue for sending is managed in FIFO order.
- TA_TPRI (0x0001)
  Task wait queue for sending is managed in task current priority order.

Note, task wait queue for receiving is managed in FIFO order.

(3) Capacity of the Data Queue Area (dtqcnt), Start Address of the Data Queue Area (dtq)

The application acquires data queue area of TSZ_DTQ(dtqcnt) bytes and specifies the start address for dtq.

Note, the µITRON 4.0 specification defines the function that the kernel allocates data queue area when NULL is specified for dtq. But RI600/PX does not support this function.

0 can be specified for dtqcnt. Since data cannot be stored in the data queue created by dtqcnt = 0, the data sending task or data receiving task that has performed its operation first will enter the WAITING state. The WAITING state of that task is canceled when the other task has performed its operation. Thus, data sending tasks and data receiving tasks are completely synchronized. Note, dtq is disregarded when dtqcnt is 0.

The kernel is not concerned of anything of the access permission to the data queue area. Usually, the data queue area should be generated to the area other than memory objects and user stacks. When the data queue area is generated in the memory object, a task with the writing permission to the memory object might rewrite data queue area by mistake.

5.10.2 Deletes Data Queue (del_dtq)

❑C Language API

```
ER        del_dtq(ID dtqid);
```

❑Parameter

```
dtqid     Data queue ID
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number dtqid<=0, VTMAX_DTQ < dtqid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The data queue indicated by dtqid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the data queue indicated by dtqid.

No error will occur even if there is waiting task for the data queue indicated by dtqid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

5.10.3 Sends to Data Queue (snd_dtq, psnd_dtq, ipsnd_dtq, tsnd_dtq, fsnd_dtq, ifsnd_dtq)

❑C Language API

```
ER        snd_dtq(ID dtqid, VP_INT data);
ER        psnd_dtq(ID dtqid, VP_INT data);
ER        ipsnd_dtq(ID dtqid, VP_INT data);
ER        tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);
ER        fsnd_dtq(ID dtqid, VP_INT data);
ER        ifsnd_dtq(ID dtqid, VP_INT data);
```

❑Parameter

dtqid	Data queue ID
data	Data to be sent
<Only for tsnd_dtq>	
tmout	Timeout (millisecond)

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	Invalid ID number dtqid<=0, VTMAX_DTQ < dtqid
E_CTX	Context error (invoked from system state not permitted)
E_ILUSE	Illegal use of service call fsnd_dtq or ifsnd_dtq is issued for the data queue which dtqcnt is 0.
E_MACV	Memory access violation (only for snd_dtq, psnd_dtq, tsnd_dtq and fsnd_dtq) Stack pointer points out of user stack for invoking task.
E_NOEXS	Non-existent object The data queue indicated by dtqid does not exist.
E_RLWAI	Forced release from the waiting (only for snd_dtq and tsnd_dtq) Accept rel_wai or irel_wai while waiting
E_TMOUT	Polling failure or timeout
E_DLT	Waiting object deleted (only for snd_dtq and tsnd_dtq) The data queue indicated by dtqid has been deleted while waiting.
EV_RST	Released from WAITING state by the object reset (vrst_dtq) (only for snd_dtq and tsnd_dtq)

❑Function

Sends the data indicated by data (4 bytes) to the data queue indicated by dtqid.

For fsnd_tdq and ifsnd_dtq, note that if a data queue created with dtqcnt = 0 is specified, the call always results in E_ILUSE error.

(1) If the target data queue contains any task waiting to receive

The data is not stored in the data queue, but instead passed to the task at the top of the receive queue, upon which the task is dequeued.

(2) If the target data queue contains no tasks waiting to receive

(a) When the data queue has space

The data is stored in the tail end of the data queue and the data queue count is incremented by 1.

(b) When the data queue has no space

1. For snd_dtq and tsnd_dtq

   The invoking task is tied to a queue (send queue) to wait for space to be created in the data queue.

   For the tsnd_dtq service call, specify a wait time in tmout. If a positive value is specified for tmout and the specified tmout time elapses while the wait condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with psnd_dtq. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. Therefore, the service call is processed in the same way as with snd_dtq.

2. For psnd_dtq and ipsnd_dtq

   An error E_TMOUT is immediately returned.

3. For fsnd_dtq, ifsnd_dtq

   Regardless of whether there is any task waiting to send, the oldest data in the data queue is deleted and data is stored into the data queue.

While one task is placed into the WAITING state by snd_dtq or tsnd_dtq, if vrst_dtq is issued from another task, then the task in the WAITING state is released from that state and the service call concerned is terminated with error EV_RST.

5.10.4 Receives from Data Queue (rcv_dtq, prcv_dtq, iprcv_dtq, trcv_dtq)

❑C Language API

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

❑Parameter

```
dtqid     Data queue ID
p_data    Pointer to the storage to which the received data is returned
<Only for trcv_dtq>
tmout     Timeout (millisecond)
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

```
E_PAR            Parameter error
                 (1) p_data == NULL
                 (2) tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID             Invalid ID number
                 dtqid<=0, VTMAX_DTQ < dtqid
E_CTX            Context error (invoked from system state not permitted)
E_MACV           Memory access violation (only for rcv_dtq, prcv_dtq and trcv_dtq)
                 (1) Stack pointer points out of user stack for invoking task.
                 (2) The operand write access to the area indicated by p_data has not been
                     permitted to the invoking task.
E_NOEXS          Non-existent object
                 The data queue indicated by dtqid does not exist.
E_RLWAI          Forced release from the waiting (only for rcv_dtq and trcv_dtq)
                 Accept rel_wai or irel_wai while waiting
E_TMOUT          Polling failure or timeout
E_DLT            Waiting object deleted (only for snd_dtq and tsnd_dtq)
                 The data queue indicated by dtqid has been deleted while waiting.
```

❑Function

Receives data from the data queue indicated by dtqid and stores the received data in the area pointed to by p_data.

If the data queue contains data, the data at the top of the queue (the oldest data) is received. When data present in the data queue is received, the data queue count is decremented by 1. In addition, if there is any task for waiting to send data, the data for the task at the top of the send queue is stored in the data queue. As a result, the task waiting to send data is released from the WAITING state.

If the data queue contains no data and there is any task waiting to send data (such a situation occurs only when the size of the data queue = 0), data for the task at the top of the data send queue is received. As a result, the task waiting to send data is released from the WAITING state.

If the data queue contains no data and there are no tasks waiting to send data either, and if the service call invoked in this situation is rcv_dtq or trcv_dtq, then the invoking task is tied to data arrival queue (receive queue); or, in the case of the prcv_dtq service call, it immediately returns error E_TMOUT. The receive queue is managed in FIFO order.

For the trcv_dtq service call, specify a wait time in tmout. If a positive value is specified for tmout and the specified tmout time elapses while the wait release condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with prcv_dtq. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. Therefore, the service call is processed in the same way as with rcv_dtq.

5.10.5 Refers to Data Queue State (ref_dtq, iref_dtq)

❑C Language API

```
ER        ref_dtq(ID dtqid, T_RDTQ *pk_rdtq);
ER        iref_dtq(ID dtqid, T_RDTQ *pk_rdtq);
```

❑Parameter

```
dtqid     Data queue ID
pk_rdtq   Pointer to the storage to which the data queue state is returned
```

❑Packet Structure

```
typedef   struct t_rdtq {
          ID      stskid;     The task ID at the top of the task wait queue to send
          ID      rtskid;     The task ID at the top of the task wait queue to receive
          UINT    sdtqcnt;    The number of data in the data queue
} T_RDTQ;
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR            Parameter error
                 pk_rdtq == NULL
E_ID             Invalid ID number
                 (1) dtqid<=0, VTMAX_DTQ < dtqid
E_CTX            Context error (invoked from system state not permitted)
                   Note : The E_CTX is not detected in the following cases.
                       (1) Invocation of ref_dtq from non-task context
                       (2) Invocation of iref_dtq from task context
E_MACV           Memory access violation (only for ref_dtq)
                 The operand write access to the area indicated by pk_rdtq has not been permitted
                 to the invoking task.
E_NOEXS          Non-existent object
                 The data queue indicated by dtqid does not exist.
```

❑Function

Refers to the state of the data queue indicated by dtqid and returns the task ID waiting to send (stskid), the task ID waiting to receive (rtskid), and the data count stored in the data queue (sdtqcnt) to the area pointed to by pk_rdtq.

If there are no tasks waiting to send and no tasks waiting to receive, TSK_NONE (= 0) is returned as the waiting task ID.

5.11 Synchronization and Communication Function (Mailbox)

Table 5.17 shows specifications of the mailbox function.

Table 5.17 Specifications of the Mailbox Function

No.	Item	Content
1	Mailbox ID	1 - VTMAX_MBX *1
2	Message priority	1 - TMAX_MPRI *2
3	Mailbox attribute	TA_TFIFO : Task wait queue is managed in FIFO order. TA_TPRI : Task wait queue is managed in task current priority order. TA_MFIFO : Message queue is managed in FIFO order. TA_MPRI : Message queue is managed in message priority order.

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates the maximum mailbox ID.

2 This is the macro output to kernel_id.h by cfg600px, which indicates the value specified for system.message_pri in the .cfg file.

Table 5.18 Service Calls for Mailbox Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_mbx		Creates mailbox	T		E	D	U	
2	acre_mbx		Creates mailbox (Automatic ID Assignment)	T		E	D	U	
3	del_mbx		Deletes mailbox	T		E	D	U	
4	snd_mbx	[S][B]	Sends to mailbox	T		E	D	U	
5	isnd_mbx				N	E	D	U	
6	rcv_mbx	[S][B]	Receives from mailbox	T		E		U	
7	prcv_mbx	[S][B]	Receives from mailbox (polling)	T		E	D	U	
8	iprcv_mbx				N	E	D	U	
9	trcv_mbx	[S]	Receives from mailbox (with timeout)	T		E		U	
10	ref_mbx		Refers to mailbox state	T		E	D	U	
11	iref_mbx				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings :

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.11.1 Creates Mailbox (cre_mbx, acre_mbx)

❑C Language API

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

❑Parameter

mbxid	Mailbox ID
pk_cmbx	Pointer to the packet containing the mailbox creation information

❑Packet Structure

```
typedef     struct t_cmbx {
            ATR     mbxatr;             Mailbox attribute
            PRI     maxmpri;            Maximum message priority
            VP      mprihd;             Start address of message queue headers for each message priority
} T_CMBX;
```

❑Return Value

In cre_mbx : E_OK for normal completion or error code
In acre_mbx : Created mailbox ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute Either of bits except bit0 and bit1 of mbxatr is 1.
E_PAR	Parameter error (1) pk_cmbx == NULL (2) maxmpri <= 0 or TMAX_MPRI < maxmpri (when TA_MPRI attribute is specified.)
E_ID	Invalid ID number (only for cre_mbx) mbxid<=0, VTMAX_MBX < mbxid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_cmbx has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOID	No ID number available (only for acre_mbx)
E_OBJ	Object state error (only for cre_mbx) The mailbox indicated by mbxid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_mbx creates a mailbox with mailbox ID indicated by mbxid according to the content of pk_cmbx. The acre_mbx creates a mailbox according to the content of pk_cmbx, and returns the created mailbox ID.

(1) Mailbox ID (mbxid)

The cre_mbx creates a mailbox with the mailbox ID indicated by mbxid.

(2) Mailbox Attribute (mbxatr)

The following are specified for mbxatr.

```
mbxatr:= ( ( TA_TFIFO || TA_TPRI ) | ( TA_MFIFO || TA_MPRI ) )
```

- TA_TFIFO (0x0000)
  Task wait queue is managed in FIFO order.
- TA_TPRI (0x0001)
  Task wait queue is managed in task current priority order.
- TA_MFIFO (0x0000)
  Message queue is managed in FIFO order.
- TA_MPRI (0x0002)
  Message queue is managed in message priority order.

(3) Start address of message queue headers for each message priority (mprihd)

mprihd is only disregarded.

5.11.2 Deletes Mailbox (del_mbx)

❑C Language API

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

❑Parameter

```
mbxid     Mailbox ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

E_ID	Invalid ID number mbxid<=0, VTMAX_MBX < mbxid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The mailbox indicated by mbxid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the mailbox indicated by mbxid.

No error will occur even if there is waiting task for the mailbox indicated by mbxid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

5.11.3 Sends to Mailbox (snd_mbx, isnd_mbx)

❑C Language API

```
ER        snd_mbx(ID mbxid, T_MSG *pk_msg);
ER        isnd_mbx(ID mbxid, T_MSG *pk_msg);
```

❑Parameter

```
mbxid     Mailbox ID
pk_msg    Start address of the message to be sent
```

❑Packet Structure

```
<Mailbox message header>
typedef   struct t_msg {
          VP      msghead;     Kernel management area
} T_MSG;
<Mailbox message header with priority>
typedef   struct t_msg_pri {
          T_MSG   msgque;      Message header
          PRI     msgpri;      Message priority
} T_MSG_PRI;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

```
E_PAR             Parameter error
                  (1) pk_msg == NULL
                  (2) msgpri<=0 or TMAX_MPRI < msgpri when the mailbox indicated by mbxid
                   has TA_MPRI attribute.
E_ID              Invalid ID number
                  (1) mbxid<=0, VTMAX_MBX < mbxid
E_CTX             Context error (invoked from system state not permitted)
E_MACV            Memory access violation (only for snd_mbx)
                  (1) Stack pointer points out of user stack for invoking task.
                  (2) The operand write access to the message header area has not been
                      permitted to the invoking task.
                      Message header area :
                         - TA_MFIFO attribute : T_MSG structure started from pk_msg
                         - TA_MPRI attribute : T_MSG_PRI structure started from pk_msg
E_NOEXS           Non-existent object
                  The eventflag indicated by flgid does not exist.
```

❑Function

Sends the message indicated by pk_msg to the mailbox indicated by mbxid.

If for the target mailbox there is already any task waiting to receive a message, the transmitted message is passed to the task at the top of the waiting queue, upon which the task is dequeued.

If there are no tasks waiting to receive a message, the message is tied to a message queue. The message queue is managed according to the attributes specified when the mailbox was created.

To send a message to a mailbox with TA_MFIFO attribute, make sure the message created has T_MSG structure added at the beginning of it, as shown in Figure 5.1.

To send a message to a mailbox with TA_MPRI attribute, make sure the message created has T_MSG_PRI structure added at the beginning of it, as shown in Figure 5.2.

The T_MSG or T_MSG_PRI area must not be rewritten after a message is transmitted because the kernel uses that area. The kernel operation cannot be guaranteed if this area is rewritten before a message is received after it is transmitted.

A message must be generated in the area that the receiver can read.

```
typedef struct {
    T_MSG   t_msg;      /* T_MSG structure                          */
    B       data[8];    /* Example of a user message data         */
                          /* structure (any structure)               */
} USER_MSG;
```

Figure 5.1 Example of a Message Format (TA_MFIFO attribute)

```
typedef struct {
    T_MSG_PRI  t_msg;    /* T_MSG_PRI strucure                      */
    B           data[8];  /* Example of a user message data        */
                          /* structure (any structure)                */
} USER_MSG;
```

Figure 5.2 Example of a Message Format (TA_MPRI attribute)

5.11.4 Receives from Mailbox (rcv_mbx, prcv_mbx, iprcv_mbx, trcv_mbx)

❑C Language API

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

❑Parameter

mbxid Mailbox ID

ppk_msg Pointer to the storage to which the start address of the received message is returned

<Only for trcv_mbx>

tmout Timeout (millisecond)

❑Packet Structure

```
<Mailbox message header>
typedef   struct t_msg {
          VP      msghead;     Kernel management area
} T_MSG;
<Mailbox message header with priority>
typedef   struct t_msg_pri {
          T_MSG   msgque;      Message header
          PRI     msgpri;      Message priority
} T_MSG_PRI;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error (1) ppk_msg == NULL (2) tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	Invalid ID number mbxid<=0, VTMAX_MBX < mbxid
E_CTX	Context error (invoked from system state not permitted) Note : The E_CTX is not detected in the following cases. (1) Invocation of prcv_mbx from non-task context (2) Invocation of iprcv_mbx from task context
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (only for rcv_mbx and trcv_mbx) (2) The operand write access to the area indicated by ppk_msg has not been permitted to the invoking task. (only for rcv_mbx, prcv_mbx and trcv_mbx)
E_NOEXS	Non-existent object The eventflag indicated by flgid does not exist.
E_RLWAI	Forced release from the waiting (only for rcv_mbx and trcv_mbx) Accept rel_wai or irel_wai while waiting
E_TMOUT	Polling failure or timeout
E_DLT	Waiting object deleted (only for rcv_mbx and trcv_mbx) The mailbox indicated by mbxid has been deleted while waiting.

❑Function

Receives a message from the mailbox indicated by mbxid and returns the received message's start address to the area pointed by ppk_msg.

If the mailbox contains no message and the service call invoked is rcv_mbx or trcv_mbx, the invoking task is tied to a message arrival queue (receive queue); or, in the case of the prcv_mbx and iprcv_mbx service calls, it immediately returns error E_TMOUT. The queue is managed according to the attributes specified when the mailbox was created.

For the trcv_mbx service call, specify a wait time in tmout. If a positive value is specified for tmout and the specified tmout time elapses while the wait release condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with prcv_mbx. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. Therefore, the service call is processed in the same way as with rcv_mbx.

5.11.5 Refers to Mailbox State (ref_mbx, iref_mbx)

❑C Language API

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

❑Parameter

mbxid	Mailbox ID
pk_rmbx	Pointer to the storage to which the mailbox state is returned

❑Packet Structure

```
(1)T_RMBX
typedef   struct t_rmbx {
          ID      wtskid;     The task ID at the top of the task wait queue
          T_MSG   *pk_msg;    Start address of the message to be received next
} T_RMBX;
(2)T_MSG
<Mailbox message header>
typedef   struct t_msg {
          VP      msghead;    Kernel management area
} T_MSG;
<Mailbox message header with priority>
typedef   struct t_msg_pri {
          T_MSG   msgque;     Message header
          PRI     msgpri;     Message priority
} T_MSG_PRI;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error pk_rmbx == NULL
E_ID	Invalid ID number (1) mbxid<=0, VTMAX_MBX < mbxid
E_CTX	Context error (invoked from system state not permitted) Note : The E_CTX is not detected in the following cases. (1) Invocation of ref_mbx from non-task context (2) Invocation of iref_mbx from task context
E_MACV	Memory access violation (only for ref_mbx) The operand write access to the area indicated by pk_rmbx has not been permitted to the invoking task
E_NOEXS	Non-existent object The mailbox indicated by mbxid does not exist.

❑Function

Refers to the state of the mailbox indicated by mbxid and returns the waiting task ID (wtskid) and the start address of the next message to be received (pk_msg) to the area pointed to by pk_rmbx.

If for the target mailbox there are no waiting tasks, TSK_NONE (= 0) is returned as the waiting task ID.

If the next message to be received is nonexistent, NULL (= 0) is returned as the message start address.

5.12 Extended Synchronization and Communication Function (Mutex)

Table 5.19 shows specifications of the mutex function.

Table 5.19 Specifications of the Mutex Function

No.	Item	Content
1	Mutex ID	1 - VTMAX_MTX *1
2	Mutex attribute	TA_CEILING : Priority ceiling protocol *2

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates the maximum mutex ID.

2 This kernel has adopted a simplified priority control rule for its TA_CEILING attribute (priority ceiling protocol). In the simplified priority control rule, although controls to heighten task priority are exercised under all circumstances, controls to lower task priority are applied only when the task concerned has no more mutexes locked (if it had multiple mutexes locked, when all of them have been freed).

Table 5.20 Service Calls for Mutex Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_mtx		Creates mutex	T		E	D	U	
2	acre_mtx		Creates mutex (Automatic ID Assignment)	T		E	D	U	
3	del_mtx		Deletes mutex	T		E	D	U	
4	loc_mtx		Locks mutex	T		E		U	
5	ploc_mtx		Locks mutex (polling)	T		E	D	U	
6	tloc_mtx		Locks mutex (with timeout)	T		E		U	
7	unl_mtx		Unlocks mutex	T		E	D	U	
8	ref_mtx		Refers to mutex state	T		E	D	U	

Notes: 1 The symbol "[S]" denotes μITRON 4.0-compliant, standard-profile service calls; "[B]" denotes μITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than μITRON 4.0 specification.

2 The letters representing the system state have the following meanings :

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.12.1 Creates Mutex (cre_mtx, acre_mtx)

❑C Language API

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

❑Parameter

mtxid	Mutex ID
pk_cmtx	Pointer to the packet containing the mutex creation information

❑Packet Structure

```
typedef     struct t_cmtx {
            ATR     mtxatr;           Mutex attribute
            PRI     ceilpri;          Ceiling priority
} T_CMTX;
```

❑Return Value

In cre_mtx : E_OK for normal completion or error code
In acre_mtx : Created mutex ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute mtxatr != TA_CEILING
E_PAR	Parameter error (1) pk_cmtx == NULL (2) ceilpri <= 0, TMAX_MPRI < ceilpri.
E_ID	Invalid ID number (only for cre_mtx) mtxid<=0, VTMAX_MTX < mtxid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_cmtx has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOID	No ID number available (only for acre_mtx)
E_OBJ	Object state error (only for cre_mtx) The mutex indicated by mtxid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_mtx creates a mutex with mutex ID indicated by mtxid according to the content of pk_cmtx. The acre_mtx creates a mutex according to the content of pk_cmtx, and returns the created mutex ID.

(1) Mutex ID (mtxid)

The cre_mtx creates a mutex with the mutex ID indicated by mtxid.

(2) Mutex Attribute (mtxatr)

Only TA_CEILING can be specified for mtxatr.

- TA_CEILING (0x0003)
  Priority ceiling protocol

(3) Ceiling Priority (ceilpri)

The current task priority of the task which locks a mutex rises to the ceilpri.

5.12.2 Deletes Mutex (del_mtx)

❑C Language API

```
ER        del_mtx(ID mtxid);
```

❑Parameter

```
mtxid     Mutex ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

E_ID	Invalid ID number mtxid<=0, VTMAX_MTX < mtxid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The mutex indicated by mtxid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the mutex indicated by mtxid.

No error will occur even if there is waiting task for the mutex indicated by mtxid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

When either of task locks the target mutex, the lock by the task is cancelled. As a result, the current task priority of the task is returned to the base priority when there is no mutex being locked by the task. The task is not notified that the mutex has been deleted. If an attempt is later made to unlock the mutex, an error E_NOEXS is returned.

5.12.3 Locks Mutex (loc_mtx, ploc_mtx, tloc_mtx)

❑C Language API

```
ER          loc_mtx(ID mtxid);
ER          ploc_mtx(ID mtxid);
ER          tloc_mtx(ID mtxid, TMO tmout);
```

❑Parameter

```
mtxid       Mutex ID
<Only for tloc_mtx>
tmout       Timeout (millisecond)
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	Invalid ID number mtxid<=0, VTMAX_MTX < mtxid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_ILUSE	Illegal use of service call (1) The mutex indicated by mtxid is already locked by the issuing task. (2) Ceiling priority violation (base priority of the invoking task > ceiling priority of the target mutex)
E_NOEXS	Non-existent object The mutex indicated by mtxid does not exist.
E_RLWAI	Forced release from the waiting (only for loc_mtx and tloc_mtx) Accept rel_wai or irel_wai while waiting
E_TMOUT	Polling failure or timeout
E_DLT	Waiting object deleted (only for rcv_mbx and trcv_mbx) The mutex indicated by mtxid has been deleted while waiting.

❑Function

Locks the mutex indicated by mtxid.

If the target mutex is not locked yet, the invoking task locks the mutex. In that case, the invoking task has its current priority raised to the ceiling priority of the mutex.

If the target mutex is already locked, the invoking task is tied to a queue, in which it is kept waiting for the mutex to become available to lock. The queue is managed in order of priority.

For the tloc_mtx service call, specify a wait time in tmout. If a positive value is specified for tmout and the specified tmout time elapses while the wait release condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with ploc_mtx. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. Therefore, the service call is processed in the same way as with loc_mtx.

5.12.4 Unlocks Mutex (unl_mtx)

❑C Language API

```
ER        unl_mtx(ID mtxid);
```

❑Parameter

mtxid	Mutex ID

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number mtxid<=0, VTMAX_MTX < mtxid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_ILUSE	Illegal use of service call The invoking task has not locked the mutex indicated by mtxid.
E_NOEXS	Non-existent object The mutex indicated by mtxid does not exist.

❑Function

Unlocks the mutex indicated by mtxid.

If there is any task waiting for the target mutex, the task at the top of the mutex queue is removed from the queue, and the task locks the mutex. At that time, the task has its current priority raised to the ceiling priority of the mutex.

If there are no tasks waiting for the mutex, the mutex is placed into an unlocked state.

This kernel has adopted a simplified priority ceiling protocol for its TA_CEILING attribute. Therefore, it is only when the invoking task has had all of its locked mutexes freed by this service call that the task's current priority is reverted to its base priority. If the invoking task still has any other mutex locked, its current priority is not changed in this service call.

5.12.5 Refers to Mutex State (ref_mtx)

❑C Language API

```
ER        ref_mtx(ID mtxid, T_RMTX *pk_rmtx);
```

❑Parameter

```
mtxid     Mutex ID
pk_rmtx   Pointer to the storage to which the mutex state is returned
```

❑Packet Structure

```
typedef   struct t_rmtx {
          ID      htskid;      Task ID locking the mutex
          ID      wtskid;      The task ID at the top of the task wait queue
} T_RMTX;
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR             Parameter error
                  pk_rmtx == NULL
E_ID              Invalid ID number
                  mtxid<=0, VTMAX_MTX < mtxid
E_CTX             Context error (invoked from system state not permitted)
E_MACV            Memory access violation
                  The operand write access to the area indicated by pk_rmbx has not been
                  permitted to the invoking task
E_NOEXS           Non-existent object
                  The mutex indicated by mtxid does not exist.
```

❑Function

Refers to the state of the mutex indicated by mtxid and returns the task ID that has the mutex locked (htskid) and the task ID at the top of the mutex queue (wtskid) to the area pointed to by pk_rmtx.

If there are no tasks that have the target mutex locked, TSK_NONE (= 0) is returned to htskid. If there are no tasks waiting for the target mutex, TSK_NONE (= 0) is returned to wtskid.

5.13 Extended Synchronization and Communication Function (Message Buffer)

Table 5.21 shows specifications of the message buffer function.

Table 5.21 Specifications of the Message Buffer Function

No.	Item	Content
1	Message buffer ID	1 - VTMAX_MBF *1
2	Buffer size	0 or 8 - 65532 (bytes)
3	Message size that can be transmitted	1 - 65528 (bytes)
4	Message buffer attribute	TA_TFIFO : Task wait queue for sending is managed in FIFO order.

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates the maximum message buffer ID.

Table 5.22 Service Calls for Message Buffer Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_mbf		Creates message buffer	T		E	D	U	
2	acre_mbf		Creates message buffer (Automatic ID Assignment)	T		E	D	U	
3	del_mbf		Deletes message buffer	T		E	D	U	
4	snd_mbf		Sends to message buffer	T		E		U	
5	psnd_mbf		Sends to message buffer (polling)	T		E	D	U	
6	ipsnd_mbf				N	E	D	U	
7	tsnd_mbf		Sends to message buffer (with timeout)	T		E		U	
8	rcv_mbf		Receives from message buffer	T		E		U	
9	prcv_mbf		Receives from message buffer (polling)	T		E	D	U	
10	trcv_mbf		Receives from message buffer (with timeout)	T		E		U	
11	ref_mbf		Refers to message buffer state	T		E	D	U	
12	iref_mbf				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings :

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.13.1 Creates Message Buffer (cre_mbf, acre_mbf)

❑C Language API

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

❑Parameter

mbfid	Message buffer ID
pk_cmbf	Pointer to the packet containing the message buffer creation information

❑Packet Structure

```
typedef     struct t_cmbf {
            ATR     mbfatr;           Message buffer attribute
            UINT    maxmsz;           Maximum message size (in bytes)
            SIZE    mbfsz;            Size of message buffer area (in bytes)
            VP      mbf;              Start address of message buffer area
} T_CMBF;
```

❑Return Value

In cre_mbf : E_OK for normal completion or error code

In acre_mbf : Created message buffer ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute mbfatr != TA_CEILING
E_PAR	Parameter error (1) pk_cmbf == NULL (2) maxmsz == 0, 65528 < maxmsz (3) 0 < mbfsz < 8, 65532 < mbfsz (4) mbfsz != 0 and mbf + mbfsz > 0x100000000. (5) mbfsz != 0 and mbfsz < maxmsz + VTSZ_MBFTBL
E_ID	Invalid ID number (only for cre_mbf) mbfid<=0, VTMAX_MBF < mbfid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_cmbf has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOMEM	Insufficient memory mbfsz!=0 and mbf == NULL
E_NOID	No ID number available (only for acre_mbf)
E_OBJ	Object state error (only for cre_mbf) The message buffer indicated by mbfid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_mbf creates a message buffer with message buffer ID indicated by mbfid according to the content of pk_cmbf. The acre_mbf creates a message buffer according to the content of pk_cmbf, and returns the created message buffer ID.

(1) Message Buffer ID (mbfid)

The cre_mbf creates a message buffer with the message buffer ID indicated by mbfid.

(2) Message Buffer Attribute (mbfatr)

Only TA_TFIFO can be specified for mbfatr.

- TA_TFIFO (0x0000)
  Task wait queue for sending is managed in FIFO order.

Note, task wait queue for receiving is managed in FIFO order.

(3) Size of message buffer area (mbfsz), Start address of message buffer area (mbf)

The application acquires message buffer area, and specifies the start address for mbf and the size for mbfsz.

Note, the µITRON 4.0 specification defines the function that the kernel allocates message buffer area when NULL is specified for mbf. But RI600/PX does not support this function.

0 can be specified for mbfsz. Since message cannot be stored in the message buffer area created by mbfsz = 0, the message sending task or message receiving task that has performed its operation first will enter the WAITING state. The WAITING state of that task is canceled when the other task has performed its operation. Thus, data sending tasks and data receiving tasks are completely synchronized. Note, mbf is disregarded when mbfsz is 0.

The kernel is not concerned of anything of the access permission to the message buffer area. Usually, the message buffer area should be generated to the area other than memory objects and user stacks. When the message buffer area is generated in the memory object, a task with the writing permission to the memory object might rewrite message buffer area by mistake.

5.13.2 Deletes Message Buffer (del_mbf)

❑C Language API

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

❑Parameter

mbfid	Message buffer ID

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number mbfid <=0, VTMAX_MBF < mbfid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The message buffer indicated by mbfid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the message buffer indicated by mbfid.

No error will occur even if there is waiting task for the message buffer indicated by mbfid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

5.13.3 Sends to Message Buffer (snd_mbf, psnd_mbf, ipsnd_mbf, tsnd_mbf)

❑C Language API

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

❑Parameter

mbfid	Message buffer ID
msg	Start address of the message to be sent
msgsz	Size of the message to be sent (in bytes)
<Only for tsnd_mbf>	
tmout	Timeout (millisecond)

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error (1) msg == NULL (2) msgsz=0, msgsz > (Maximum message size specified at creation) (3) tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	Invalid ID number mbfid<=0, VTMAX_MBF < mbfid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for snd_mbf, psnd_mbf and tsnd_mbf) (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the message area (start address = msg, size = msgsz) has not been permitted to the invoking task.
E_NOEXS	Non-existent object The message buffer indicated by mbfid does not exist.
E_RLWAI	Forced release from the waiting (only for snd_mbf and tsnd_mbf) Accept rel_wai or irel_wai while waiting
E_TMOUT	Polling failure or timeout
E_DLT	Waiting object deleted (only for rcv_mbf and trcv_mbf) The message buffer indicated by mbfid has been deleted while waiting.
EV_RST	Released from WAITING state by the object reset (vrst_mbf) (only for snd_mbf and tsnd_mbf)

❑Function

Sends the message indicated by msg to the message buffer indicated by mbfid. The transmitted size is indicated in bytes by msgsz.

If for the target message buffer there is any task waiting to receive, the message is not stored in the message buffer, but instead passed to the task at the top of the receive queue, upon which the task is dequeued.

If for the target message buffer there is already any task waiting to send and the service call concerned is snd_mbf or tsnd_mbf, then the task is tied to a queue (send queue) in which it is kept waiting for space in the message buffer to become available; or, in the case of the psnd_mbf and ipsnd_mbf service calls, it immediately returnes error E_TMOUT. The send queue is arranged in FIFO order.

If there are no tasks waiting to receive or no tasks waiting to send, the message is stored in the message buffer. As a result, the size of free space in the message buffer is reduced by an amount that is calculated by the equation below.

- Reduced size = (msgsz rounded up to a multiple of 4) + VTSZ_MBFTBL

Here, VTSZ_MBFTBL means the size of the management table (4-bytes) which is generated in the buffer area by the kernel.

If the message buffer does not have as much space as this size (including the case where the buffer size = 0) and the service call concerned is snd_mbf or tsnd_mbf, then the invoking task is tied to a send queue; or, for psnd_mbf and ipsnd_mbf, it immediately returns error E_TMOUT.

For the tsnd_mbf service call, specify a wait time in tmout.

If a positive value is specified for tmout and the specified tmout time elapses while the wait condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with psnd_mbf. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. Therefore, the service call is processed in the same way as with snd_mbf.

When the task placed in the top of wait queue to send is release from the WAITING state by rel_wai or irel_wai service call or is terminated by ter_tsk service call, other waiting tasks to send might be release from the WAITING state.

While one task is placed into the WAITING state by snd_mbf or tsnd_mbf, if vrst_mbf is issued from another task, then the task in the WAITING state is released from that state and the service call concerned is terminated with error EV_RST.

5.13.4 Receives from Message Buffer (rcv_mbf, prcv_mbf, trcv_mbf)

❑C Language API

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

❑Parameter

mbfid	Message buffer ID
msg	Pointer to the storage to which the received message is stored
<Only for trcv_mbf>	
tmout	Timeout (millisecond)

❑Return Value

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

❑Error Code

E_PAR	Parameter error (1) msg == NULL (2) tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	Invalid ID number mbfid<=0, VTMAX_MBF < mbfid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the message area (start address = msg, size = maximum message size specified at creation) has not been permitted to the invoking task
E_NOEXS	Non-existent object The message buffer indicated by mbfid does not exist.
E_RLWAI	Forced release from the waiting (only for rcv_mbf and trcv_mbf) Accept rel_wai or irel_wai while waiting
E_TMOUT	Polling failure or timeout
E_DLT	Waiting object deleted (only for rcv_mbf and trcv_mbf) The message buffer indicated by mbfid has been deleted while waiting.

❑Function

Receives a message from the message buffer indicated by mbfid and stores the received message in the area pointed to by msg. Also, the size of the received message is returned as return parameter.

If the message buffer contains messages, the one at the top of the message queue (the oldest message) is received. When messages present in the message buffer are received this way, the size of free space in the message buffer is increased by an amount calculated by the equation below.

- Increased size = (msgsz rounded up to a multiple of 4) + VTSZ_MBFTBL

Here, VTSZ_MBFTBL means the size of the management table (4-bytes) which is generated in the buffer area by the kernel.

As a result, if the message buffer now has a free space larger than the size of the message that the task at the top of the message send queue was trying to send, then the message is stored in the message buffer, upon which the task is dequeued. If the message buffer is large enough to store messages for the other tasks in the send queue, the kernel processes in the same way in the order the message send queue.

If the message buffer size = 0 and there is any task waiting to send, a message for the task at the top of the send queue is received. As a result, the task kept waiting to send a message is dequeued.

If the message buffer contains no messages and there are no tasks waiting to send a message either, and if the service call invoked is rcv_mbf or trcv_mbf, then the invoking task is tied to a queue (receive queue) in which it is kept waiting for a message to arrive; or, in the case of the prcv_mbf service call, it immediately returns error E_TMOUT. The receive queue is managed in FIFO order.

For the trcv_mbf service call, specify a wait time in tmout.

If a positive value is specified for tmout and the specified tmout time elapses while the wait release condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with prcv_mbf. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. Therefore, the service call is processed in the same way as with rcv_mbf.

5.13.5 Refers to Message Buffer State (ref_mbf, iref_mbf)

❑C Language API

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

❑Parameter

mbfid	Message Buffer ID
pk_rmbf	Pointer to the storage to which the message buffer state is returned

❑Packet Structure

```
typedef   struct t_rmbf {
          ID      stskid;      The task ID at the top of the task wait queue to send
          ID      rtskid;      The task ID at the top of the task wait queue to receive
          UINT    smsgcnt;     The number of messages in the message buffer
          SIZE    fmbfsz;      Size of free message buffer area (in bytes)
} T_RMBF;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error pk_rmbf == NULL
E_ID	Invalid ID number mbfid<=0, VTMAX_MBF < mbfid
E_CTX	Context error (invoked from system state not permitted) Note : The E_CTX is not detected in the following cases. (1) Invocation of ref_mbf from non-task context (2) Invocation of iref_mbf from task context
E_MACV	Memory access violation (only for ref_mbf) The operand write access to the area indicated by pk_rmbf has not been permitted to the invoking task
E_NOEXS	Non-existent object The message buffer indicated by mbfid does not exist.

❑Function

Refers to the state of the message buffer indicated by mbfid and returns the task ID waiting to send (stskid), the task ID waiting to receive (rtskid), the number of messages contained in the message buffer (smsgcnt), and the free buffer size (fmbfsz) to the area pointed to by pk_rmbf.

If there are no tasks waiting to receive or no tasks waiting to send, TSK_NONE (= 0) is returned as the waiting task ID.

5.14 Memory Pool management Function (Fixed-sized Memory Pool)

Table 5.23 shows specifications of the fixed-sized memory pool function.

Table 5.23 Specifications of the Fixed-sized Memory Pool Function

No.	Item	Content
1	Fixed-sized memory pool ID	1 - VTMAX_MPF *1
2	Maximum number of memory blocks	65535
3	Maximum size of memory block	65535 (bytes)
4	Maximum pool size (block size × number of blocks)	VTMAX_AREASIZE (bytes) *2
5	Fixed-sized memory pool attribute	TA_TFIFO : Task wait queue is managed in FIFO order. TA_TPRI : Task wait queue is managed in task current priority order.

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates the maximum fixed-sized memory pool ID.

2 This is the macro defined in kernel.h. The definition is 256 M-bytes.

Table 5.24 Service Calls for Fixed-sized Memory Pool Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_mpf		Creates fixed-sized memory pool	T		E	D	U	
2	acre_mpf		Creates fixed-sized memory pool (Automatic ID Assignment)	T		E	D	U	
3	del_mpf		Deletes fixed-sized memory pool	T		E	D	U	
4	get_mpf	[S][B]	Acquires fixed-sized memory block	T		E		U	
5	pget_mpf	[S][B]	Acquires fixed-sized memory block (polling)	T		E	D	U	
6	ipget_mpf				N	E	D	U	
7	tget_mpf	[S]	Acquires fixed-sized memory block (with timeout)	T		E		U	
8	rel_mpf	[S][B]	Releases fixed-sized memory block	T		E	D	U	
9	irel_mpf				N	E	D	U	
10	ref_mpf		Refers to fixed-sized memory pool state	T		E	D	U	
11	iref_mpf				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.14.1 Creates Fixed-sized Memory Pool (cre_mpf, acre_mpf)

❑C Language API

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

❑Parameter

mpfid	Fixed-sized memory pool ID
pk_cmpf	Pointer to the packet containing the fixed-sized memory pool creation information

❑Packet Structure

```
typedef     struct t_cmpf {
            ATR     mpfatr;            Fixed-sized memory pool attribute
            UINT    blkcnt;            Total number of memory blocks
            UINT    blksz;             Memory block size (in bytes)
            VP      mpf;               Start address of fixed-sized memory pool area
            VP      mpfmb;             Start address of fixed-sized memory pool management area
} T_CMPF;
```

❑Return Value

In cre_mpf : E_OK for normal completion or error code

In acre_mpf : Created fixed-sized memory pool ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute Either of bits except bit0 of mpfatr is 1.
E_PAR	Parameter error (1) pk_cmpf == NULL (2) blksz == 0, 65535 < blksz (3) blkcnt == 0, 65535 < blkcnt (4) TSZ_MPF(blkcnt, blksz) > VTMAX_AREASIZE (5) mpf + TSZ_MPF(blkcnt, blksz) > 0x100000000
E_ID	Invalid ID number (only for cre_mpf) mpfid<=0, VTMAX_MPF < mpfid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_cmpf has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOMEM	Insufficient memory (1) mpf == NULL (2) mpfmb == NULL
E_NOID	No ID number available (only for acre_mpf)
E_OBJ	Object state error (only for cre_mpf) The fixed-sized memory pool indicated by mpfid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_mpf creates a fixed-sized memory pool with fixed-sized memory pool ID indicated by mpfid according to the content of pk_cmpf. The acre_mpf creates a fixed-sized memory pool according to the content of pk_cmpf, and returns the created fixed-sized memory pool ID.

(1) Fixed-sized Memory Pool ID (mpfid)

The cre_mpf creates a fixed-sized memory pool with the fixed-sized memory pool ID indicated by mpfid.

(2) Fixed-sized Memory Pool Attribute (mpfatr)

The following are specified for mpfatr.

```
mpfatr:= ( TA_TFIFO || TA_TPRI )
```

- TA_TFIFO (0x0000)
  Task wait queue is managed in FIFO order.
- TA_TPRI (0x0001)
  Task wait queue is managed in task current priority order.

(3) Total Number of Memory Blocks (blkcnt), Memory Block Size (blksz), Start Address of Fixed-sized Memory Pool Area (mpf)

The application acquires fixed-sized memory pool area of TSZ_MPF(blkcnt, blksz) bytes and specifies the start address for mpf.

Note, the µITRON 4.0 specification defines the function that the kernel allocates fixed-sized memory pool area when NULL is specified for mpf. But RI600/PX does not support this function.

The kernel is not concerned of anything of the access permission to the fixed-sized memory pool area. To access to the memory block from task, the memory pool area must be in the memory object with appropriate permission.

Note, the kernel generates management tables in the memory pool area. If the management table is rewritten by allocation, correct system operation cannot be guaranteed.

(4) Start Address of Fixed-sized Memory Pool Management Area (mpfmb)

The application acquires fixed-sized memory pool management area of TSZ_MPFMB(blkcnt, blksz) bytes and specifies the start address for mpfmb.

The kernel is not concerned of anything of the access permission to the fixed-sized memory pool management area. Usually, the fixed-sized memory pool management area should be generated to the area other than memory objects and user stacks. When the fixed-sized memory pool management area is generated in the memory object, a task with the writing permission to the memory object might rewrite the fixed-sized memory pool management area by mistake.

5.14.2 Deletes Fixed-sized Memory Pool (del_mpf)

❑C Language API

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

❑Parameter

```
mpfid     Fixed-sized memory pool ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

E_ID	Invalid ID number mpfid <=0, VTMAX_MPF < mpfid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The fixed-sized memory pool indicated by mpfid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the fixed-sized memory pool indicated by mpfid.

No error will occur even if there is waiting task for the fixed-sized memory pool indicated by mpfid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

5.14.3 Acquires fixed-sized memory block (get_mpf, pget_mpf, ipget_mpf, tget_mpf)

❑C Language API

```
ER        get_mpf(ID mpfid, VP *p_blk);
ER        pget_mpf(ID mpfid, VP *p_blk);
ER        ipget_mpf(ID mpfid, VP *p_blk);
ER        tget_mpf(ID mpfid, VP *p_blk, TMO tmout);
```

❑Parameter

```
mpfid     Fixed-sized memory pool ID
p_blk     Pointer to the storage to which the start address of the memory block
          is returned
<Only for tget_mpf>
tmout     Timeout (millisecond)
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

```
E_PAR              Parameter error
                   (1) p_blk == NULL
                   (2) tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID               Invalid ID number
                   mpfid<=0, VTMAX_MPF < mpfid
E_CTX              Context error (invoked from system state not permitted)
                     Note: The E_CTX is not detected in the following cases.
                         (1) Invocation of pget_mpf from non-task context
                         (2) Invocation of ipget_mpf from task context
E_MACV             Memory access violation
                   (1) Stack pointer points out of user stack for invoking task.
                       (only for get_mpf and tget_mpf)
                   (2) The operand write access to the area indicated by p_blk has not been
                       permitted to the invoking task. (only for get_mpf, pget_mpf and tget_mpf)
E_NOEXS            Non-existent object
                   The fixed-sized memory pool indicated by mpfid does not exist.
E_RLWAI            Forced release from the waiting (only for get_mpf and tget_mpf)
                   Accept rel_wai or irel_wai while waiting
E_TMOUT            Polling failure or timeout
E_DLT              Waiting object deleted (only for get_mpf and tget_mpf)
                   The fixed-sized memory pool indicated by mpfid has been deleted while waiting.
EV_RST             Released from WAITING state by the object reset (vrst_mpf)
                   (only for get_mpf and tget_mpf)
```

❑Function

Acquires one memory block from the fixed-sized memory pool indicated by mpfid and returns the start address of the acquired memory block to the area pointed to by p_blk.

If there is already any task waiting to acquire a memory block, or if there are no waiting tasks but the target fixed-sized memory pool has no free blocks in it, and the service call invoked is get_mpf or tget_mpf, then the invoking task is tied to a memory pool memory acquisition queue; or, in the case of the pget_mpf and ipget_mpf service calls, it immediately returns error E_TMOUT. The queue is managed according to the attributes specified when the fixed-sized memory pool was created.
For the tget_mpf service call, specify a wait time in tmout. If a positive value is specified for tmout and the specified tmout time elapses while the wait release condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with pget_mpf. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. Therefore, the service call is processed in the same way as with get_mpf.

While one task is placed into the WAITING state by get_mpf or tget_mpf, if vrst_mpf is issued from another task, then the task in the WAITING state is released from that state and the service call concerned is terminated with error EV_RST.

❑Supplement

The address boundary adjustment number for the memory blocks acquired is 1.

If memory blocks need to be acquired with a larger boundary address than that, observe the following at creating the fixed-sized memory pool:

(1) Specify the memory block size to a multiple of the desired boundary adjustment number.
(2) Specify the desired boundary adjustment number for start address of the memory pool area.

5.14.4 Releases Fixed-sized Memory Pool (rel_mpf, irel_mpf)

❑C Language API

```
ER          rel_mpf(ID mpfid, VP blk);
ER          irel_mpf(ID mpfid, VP blk);
```

❑Parameter

mpfid	Fixed-sized memory pool ID
blk	Start address of the memory block to be released

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error (1) blk == NULL (2) blk is the address other than the memory block, or the address of the memory block which has been released
E_ID	Invalid ID number mpfid<=0, VTMAX_MPF < mpfid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for rel_mpf) Stack pointer points out of user stack for invoking task.
E_NOEXS	Non-existent object The fixed-sized memory pool indicated by mpfid does not exist.

❑Function

Releases the memory block indicated by blk to the fixed-sized memory pool indicated by mpfid.

For blk, specify the start address of the memory block acquired by the get_mpf, pget_mpf, ipget_mpf, or tget_mpf service call.

If for the target fixed-sized memory pool there is any task waiting to acquire a memory block, then the block returned by the service call concerned is assigned to the task at the top of the waiting queue, upon which the task is dequeued.

5.14.5 Refers to Fixed-sized Memory Pool State (ref_mpf, iref_mpf)

❑C Language API

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

❑Parameter

mpfid	Fixed-sized memory pool ID
pk_rmpf	Pointer to the storage to which the fixed-sized memory pool state is returned

❑Packet Structure

```
typedef   struct t_rmpf {
          ID      wtskid;      The task ID at the top of the task wait queue
          UINT    fblkcnt;     The number of free memory blocks
} T_RMPF;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error pk_rmpf == NULL
E_ID	Invalid ID number mpfid<=0, VTMAX_MPF < mpfid
E_CTX	Context error (invoked from system state not permitted) Note: The E_CTX is not detected in the following cases. (1) Invocation of ref_mpf from non-task context (2) Invocation of iref_mpf from task context
E_MACV	Memory access violation (only for ref_mpf) The operand write access to the area indicated by pk_rmpf has not been permitted to the invoking task.
E_NOEXS	Non-existent object The fixed-sized memory pool indicated by mpfid does not exist.

❑Function

Refers to the state of the fixed-sized memory pool indicated by mpfid and returns the waiting task ID (wtskid) and the number of free blocks (fblkcnt) to the area pointed to by pk_rmpf.

If for the target memory pool there are no waiting tasks, TSK_NONE (= 0) is returned as the waiting task ID.

5.15 Memory Pool management Function (Variable-sized Memory Pool)

Table 5.25 shows specifications of the variable-sized memory pool function.

Table 5.25 Specifications of the Variable-sized Memory Pool Function

No.	Item	Content
1	Variable-sized memory pool ID	1 - VTMAX_MPL *1
2	Pool size	24 - VTMAX_AREASIZE (bytes) *2
3	Block size	1- 0xBFFFFF4 (bytes)
4	Variable-sized memory pool attribute	TA_TFIFO : Task wait queue is managed in FIFO order.

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates the maximum variable-sized memory pool ID.

2 This is the macro defined in kernel.h. The definition is 256 M-bytes.

Table 5.26 Service Calls for Variable-sized Memory Pool Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_mpl		Creates variable-sized memory pool	T		E	D	U	
2	acre_mpl		Creates variable-sized memory pool (atomatic ID asignment)	T		E	D	U	
3	del_mpl		Deletes variable-sized memory pool	T		E	D	U	
4	get_mpl		Acquires variable-sized memory block	T		E		U	
5	pget_mpl		Acquires variable-sized memory block (polling)	T		E	D	U	
6	ipget_mpl				N	E	D	U	
7	tget_mpl		Acquires variable-sized memory block (with timeout)	T		E		U	
8	rel_mpl		Releases variable-sized memory block	T		E	D	U	
9	ref_mpl		Refers to variable-sized memory pool state	T		E	D	U	
10	iref_mpl				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.15.1 Creates Variable-sized Memory Pool (cre_mpl, acre_mpl)

❑C Language API

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

❑Parameter

mplid	Variable-sized memory pool ID
pk_cmpl	Pointer to the packet containing the variable-sized memory pool creation information

❑Packet Structure

```
typedef   struct t_cmpl {
          ATR     mplatr;          Variable-sized memory pool attribute
          SIZE    mplsz;           Size of variable-sized memory pool area
          VP      mpl;             Start address of the variable-sized memory pool area
          UINT    maxblksz;        Maximum memory block size
} T_CMPL;
```

❑Return Value

In cre_mpl : E_OK for normal completion or error code
In acre_mpl : Created variable-sized memory pool ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute mplatr != TA_TFIFO.
E_PAR	Parameter error (1) pk_cmpl == NULL (2) mplsz < 24, VTMAX_AREASIZE < mplsz (3) maxblksz == 0, 0x0BFFFFF4 < maxblksz (4) mplsz is too small to maxblksz. (5) mpl + mplsz > 0x100000000 (6) mpl is not 4-bytes boundary.
E_ID	Invalid ID number (only for cre_mpl) mplid<=0, VTMAX_MPL < mplid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_cmpl has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOMEM	Insufficient memory mpl == NULL
E_NOID	No ID number available (only for acre_mpl)
E_OBJ	Object state error (only for cre_mpl) The variable-sized memory pool indicated by mplid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_mpl creates a variable-sized memory pool with variable-sized memory pool ID indicated by mplid according to the content of pk_cmpl. The acre_mpl creates a variable-sized memory pool according to the content of pk_cmpl, and returns the created variable-sized memory pool ID.

(1) Variable-sized memory pool ID (mplid)

The cre_mpl creates a variable-sized memory pool with the variable-sized memory pool ID indicated by mplid.

(2) Variable-sized memory pool Attribute (mplatr)

Only TA_TFIFO can be specified for mplatr.

- TA_TFIFO (0x0000)
  Task wait queue is managed in FIFO order.

(3) Size of variable-sized memory pool area (mplsz), Start address of the variable-sized memory pool area (mpl)

The application acquires variable-sized memory pool area of mplsz bytes and specifies the start address for mpl.

Note, the μITRON 4.0 specification defines the function that the kernel allocates variable-sized memory pool area when NULL is specified for mpl. But RI600/PX does not support this function.

The kernel is not concerned of anything of the access permission to the variable-sized memory pool area. To access to the memory block from task, the memory pool area must be in the memory object with appropriate permission.

Note, the kernel generates management tables in the memory pool area. If the management table is rewritten by allocation, correct system operation cannot be guaranteed.

(4) Maximum memory block size (maxblksz)

Specify the maximum memory block size. Note, the maximum size of memory block that can be actually acquired might be larger than maxblksz.

In the current implementation of the kernel, the size of memory block actually acquired is selected from 12 kinds of variation in the maximum. This variation is decided according to maxblksz. Table 5.27 shows variation of memory block size. Note, this behavior will be subjected to change in the future.

Table 5.27 Variation of Memory Block Size

No.	Memory block size (in hexadecimal)	Example-1 : maxblksz == 0x100	Example-2 : maxblksz == 0x20000
1	12 (0xC)	Used	-
2	36 (0x24)	Used	-
3	84 (0x54)	Used	Used
4	180 (0xB4)	Used	Used
5	372 (0x174)	-	Used
6	756 (0x2F4)	-	Used
7	1524 (0x5F4)	-	Used
8	3060 (0xBF4)	-	Used
9	6132 (0x17F4)	-	Used
10	12276 (0x2FF4)	-	Used
11	24564 (0x5FF4)	-	Used
12	49140 (0xBFF4)	-	Used
13	98292 (0x17FF4)	-	Used
14	196596 (0x2FFF4)	-	Used
15	393204 (0x5FFF4)	-	-
16	786420 (0xBFFF4)	-	-
17	1572852 (0x17FFF4)	-	-
18	3145716 (0x2FFFF4)	-	-
19	6291444 (0x5FFFF4)	-	-
20	12582900 (0xBFFFF4)	-	-
21	25165812 (0x17FFFF4)	-	-
22	50331636 (0x2FFFFF4)	-	-
23	100663284 (0x5FFFFF4)	-	-
24	201326580 (0xBFFFFF4)	-	-

5.15.2 Deletes Variable-sized Memory Pool (del_mpl)

❑C Language API

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

❑Parameter

```
mplid     Variable-sized memory pool ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

E_ID	Invalid ID number mplid <=0, VTMAX_MPL < mplid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The variable-sized memory pool indicated by mplid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the variable-sized memory pool indicated by mplid.

No error will occur even if there is waiting task for the variable-sized memory pool indicated by mplid. However, in that case, the task in the WAITING state will be released and error code E_DLT will be returned.

5.15.3 Acquires variable-sized Memory Block (get_mpl, tget_mpl, pget_mpl, ipget_mpl)

❑C Language API

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

❑Parameter

mplid	Variable-sized memory pool ID
blksz	Memory block size (in bytes)
p_blk	Pointer to the storage to which the start address of the memory block is returned
<Only for tget_mpl>	
tmout	Timeout (millisecond)

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error (1) p_blk == NULL (2) blksz == 0 (3) blksz exceeds the maximum size that can be acquired. (4) tmout < -1, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < tmout
E_ID	Invalid ID number mplid<=0, VTMAX_MPL < mplid
E_CTX	Context error (invoked from system state not permitted) Note: The E_CTX is not detected in the following cases. (1) Invocation of pget_mpl from non-task context (2) Invocation of ipget_mpl from task context
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (only for get_mpl and tget_mpl) (2) The operand write access to the area indicated by p_blk has not been permitted to the invoking task. (only for get_mpl, pget_mpl and tget_mpl)
E_NOEXS	Non-existent object The variable-sized memory pool indicated by mplid does not exist.
E_RLWAI	Forced release from the waiting (only for get_mpl and tget_mpl) Accept rel_wai or irel_wai while waiting
E_TMOUT	Polling failure or timeout
E_DLT	Waiting object deleted (only for get_mpl and tget_mpl) The variable-sized memory pool indicated by mplid has been deleted while waiting.
EV_RST	Released from WAITING state by the object reset (vrst_mpl) (only for get_mpl and tget_mpl)

❑Function

Acquires a memory block of a size (in bytes) equal to or greater than blksz from the variable-sized memory pool indicated by mplid and returns the start address of the acquired memory block to the area pointed to by p_blk.

If there is already any task waiting to acquire a memory block and the service call invoked is get_mpl or tget_mpl, then the invoking task is tied to a memory acquisition queue, in which it is kept waiting for memory acquisition; or, in the case of the pget_mpl and ipget_mpl service calls, it immediately returns error E_TMOUT. The memory acquisition queue is managed in FIFO order.

For the tget_mpl service call, specify a wait time in tmout.

If a positive value is specified for tmout and the specified tmout time elapses while the wait condition remains unmet, E_TMOUT is returned as error code. If tmout = TMO_POL (= 0) is specified, the service call is processed in the same way as with pget_mpl. If tmout = TMO_FEVR (= -1) is specified, time-outs are not watched. Therefore, the service call is processed in the same way as with get_mpl.

If the task at the top of a variable-sized memory pool memory acquisition queue is removed from the queue by the rel_wai or irel_wai service call or forcibly terminated by the ter_tsk service call, it is possible that other tasks kept waiting to acquire variable-sized memory pool memory blocks will be released from the wait state.

While one task is placed into the WAITING state by get_mpl or tget_mpl, if vrst_mpl is issued from another task, then the task in the WAITING state is released from that state and the service call concerned is terminated with error EV_RST.

❑Supplement

The address boundary adjustment number for the memory blocks acquired is different depending on the method of creating memory pool.

- Created by using cre_mpl or acre_mpl
  The address boundary adjustment number for the memory blocks acquired is 4.
- Created by cfg file
  The address boundary adjustment number for the memory blocks acquired is 1.
  However, the address boundary adjustment number for the memory blocks acquired is 4 when all variable-sized memory pool areas are divided to unique sections and the sections are located to 4-bytes boundary address at linking..

5.15.4 Release Variable-sized Memory Block (rel_mpl)

❑C Language API

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

❑Parameter

```
mplid     Variable-sized memory pool ID
blk       Start address of the memory block to be released
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR              Parameter error
                   (1) blk == NULL
                   (2) blk is the address other than the memory block, or the address
                       of the memory block which has been released
E_ID               Invalid ID number
                   mplid<=0, VTMAX_MPL < mplid
E_CTX              Context error (invoked from system state not permitted)
E_MACV             Memory access violation
                   Stack pointer points out of user stack for invoking task.
E_NOEXS            Non-existent object
                   The variable-sized memory pool indicated by mplid does not exist.
```

❑Function

Returns the memory block indicated by blk to the variable-sized memory pool indicated by mplid.

For blk, specify the start address of the memory block acquired by the get_mpl, pget_mpl, ipget_mpl, or tget_mpl service call.

When memory blocks are returned, the size of free space in a variable-sized memory pool increases. As a result, if the target variable-sized memory pool has generated in it a contiguous free space that is just as large as requested by the task at the top of the memory block acquisition queue, then the task is assigned a memory block, upon which the task is dequeued. If there are memory blocks allocatable to the other tasks in the queue, the kernel processes in the same way in the order the memory block acquisition queue.

5.15.5 Refers to Variable-sized Memory Pool State (ref_mpl, iref_mpl)

❑C Language API

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

❑Parameter

mplid	Variable-sized memory pool ID
pk_rmpl	Pointer to the storage to which the variable-sized memory pool state is returned

❑Packet Structure

```
typedef    struct t_rmpl {
           ID      wtskid;     The task ID at the top of the task wait queue
           SIZE    fmplsz;     Total size of free memory blocks (in bytes)
           UINT    fblksz;     Maximum memory block size available (in bytes)
} T_RMPL;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error pk_rmpl == NULL
E_ID	Invalid ID number mplid<=0, VTMAX_MPL < mplid
E_CTX	Context error (invoked from system state not permitted) Note : The E_CTX is not detected in the following cases. (1) Invocation of ref_mpl from non-task context (2) Invocation of iref_mpl from task context
E_MACV	Memory access violation (only for ref_mpl) The operand write access to the area indicated by pk_rmpl has not been permitted to the invoking task.
E_NOEXS	Non-existent object The variable-sized memory pool indicated by mplid does not exist.

❑Function

Refers to the state of the variable-sized memory pool indicated by mplid and returns the waiting task ID (wtskid), the total size of currently free space (fmplsz), and the largest acquirable size of memory block (fblksz) to the area pointed to by pk_rmpl.

Normally, a free space is divided up into smaller areas, so that the maximum size returned to fblksz is that of one contiguous area in a divided space. A memory block in size of up to fblksz can be acquired immediately by only invoking the pget_mpl service call once.

5.16 Time Management Function (System Time)

Table 5.28 shows specifications of the system time management.

Table 5.28 Specifications of the System Time Management Function

No.	Item	Content
1	System Time	Unsigned 48 bits
2	Units of system time	1 (millisecond)
3	Update cycle of system time	TIC_NUME/TIC_DENO (millisecond) *1
4	Initial value of system time	0x000000000000
5	Maximum value of system time	0xFFFFFFFFFFFF

Notes: 1 TIC_NUME and TIC_DENO are the macros output to kernel_id.h by cfg600px, which respectively denote the numerator (system.tic_nume) and the denominator (system.tic_deno) of the time tick cycle defined in the cfg file.

Table 5.29 Service Calls for System Time Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	set_tim	[S]	Sets system time	T		E	D	U	
2	iset_tim				N	E	D	U	
3	get_tim	[S]	Refers to system time	T		E	D	U	
4	iget_tim				N	E	D	U	
5	isig_tim	[S]	Supplies time tick	(Automatically executed by specifying CMT0, CMT1, CMT2, or CMT3 for clock.timer in the cfg file)					

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.16.1 Sets System Time (set_tim, iset_tim)

❑C Language API

```
ER        set_tim(SYSTIM *p_systim);
ER        iset_tim(SYSTIM *p_systim);
```

❑Parameter

p_systim   Pointer to the packet containing the system time to be set

❑Packet Structure

```
typedef   struct systim {
          UH      utime;      System time (upper)
          UW      ltime;      System time (lower)
} SYSTIM;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error p_systim == NULL
E_CTX	Context error (invoked from system state not permitted) Note: The E_CTX is not detected in the following cases. (1) Invocation of set_tim from non-task context (2) Invocation of iset_tim from task context
E_MACV	Memory access violation (only for set_tim) The operand read access to the area indicated by p_systim has not been permitted to the invoking task.

❑Function

Sets the current time of day that the system keeps to the value indicated by p_systim.

Note that even if the system time is changed, the actual time of day at which the time management requests made before that (e.g., task timeouts, task delay by dly_tsk, cyclic handlers, and alarm handlers) are generated will not change.

5.16.2 Refer to System Time (get_tim, iget_tim)

❑C Language API

```
ER        get_tim(SYSTIM *p_systim);
ER        iget_tim(SYSTIM *p_systim);
```

❑Parameter

```
p_systim  Pointer to the storage to which the system time is returned
```

❑Packet Structure

```
typedef   struct systim {
          UH      utime;      System time (upper)
          UW      ltime;      System time (lower)
} SYSTIM;
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR             Parameter error
                  p_systim == NULL
E_CTX             Context error (invoked from system state not permitted)
                    Note: The E_CTX is not detected in the following cases.
                        (1) Invocation of get_tim from non-task context
                        (2) Invocation of iget_tim from task context
E_MACV            Memory access violation (only for get_tim)
                  The operand write access to the area indicated by p_systim has not been
                  permitted to the invoking task.
```

❑Function

Reads out the current value of the system time and returns the result to the area pointed to by p_systim.

5.16.3 Supplies Time Tick (isig_tim)

❑Function

Updates the system time.

If either CMT0, CMT1, CMT2 or CMT3 is specified for clock.timer in the cfg file, the system is configured in such a way that processing equivalent of the isig_tim service call is automatically executed in cycles calculated by TIC_NUME / TIC_DENO (millisecond). In other words, this function is not a service call and, therefore, does not need to be invoked from the application.

When a time tick is supplied, the kernel performs the following time-related processing:

(1) Updates the system time
(2) Activates time event handler
(3) Performs time-out processing on tasks placed in the WAITING state by tslp_tsk or other service call with time-outs included

5.17 Time Management Function (Cyclic Handler)

Table 5.30 shows specifications of the cyclic handler function.

Table 5.30 Specifications of the Cyclic Handler Function

No.	Item	Content
1	Cyclic handler ID	1 - VTMAX_CYC *1
2	Activation cycle	1 - (0x7FFFFFFF - TIC_NUME)/TIC_DENO *2
3	Activation phase	0 - (0x7FFFFFFF - TIC_NUME)/TIC_DENO *2
4	Extension information (parameters passed to handler)	32 bits
5	Cyclic handler attribute	TA_HLNG : Written in high-level language TA_STA : Cyclic handler is an operational state after creation TA_PHS : Cyclic handler is activated preserving the activation phase

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates the maximum cyclic handler ID.

2 TIC_NUME and TIC_DENO are the macros output to kernel_id.h by cfg600px, which respectively denote the numerator (system.tic_nume) and the denominator (system.tic_deno) of the time tick cycle defined in the cfg file.

Table 5.31 Service Calls for Cyclic Handler Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_cyc		Creates cyclic handler	T		E	D	U	
2	acre_cyc		Creates cyclic handler (automatic ID assignment)	T		E	D	U	
3	del_cyc		Deletes cyclic handler	T		E	D	U	
4	sta_cyc	[S][B]	Starts cyclic handler operation	T		E	D	U	
5	ista_cyc				N	E	D	U	
6	stp_cyc	[S][B]	Stops cyclic handler operation	T		E	D	U	
7	istp_cyc				N	E	D	U	
8	ref_cyc		Refers to cyclic handler state	T		E	D	U	
9	iref_cyc				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.17.1 Creates Cyclic Handler (cre_cyc, acre_cyc)

❑C Language API

```
ER          cre_cyc(ID cycid, T_CCYC *pk_ccyc);
ER_ID       acre_cyc(T_CCYC *pk_ccyc);
```

❑Parameter

cycid	Cyclic handler ID
pk_ccyc	Pointer to the packet containing the cyclic handler creation information

❑Packet Structure

```
typedef     struct t_ccyc {
            ATR     cycatr;          Cyclic handler attribute
            VP_INT  exinf;           Extended information
            FP      cychdr;          Cyclic handler start address
            RELTIM  cyctim;          Activation cycle
            RELTIM  cycphs;          Activation phase
} T_CCYC;
```

❑Return Value

In cre_cyc : E_OK for normal completion or error code

In acre_cyc : Created cyclic handler ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute Either of bits except bit1 and bit2 of cycatr is 1.
E_PAR	Parameter error (1) pk_ccyc == NULL (2) cychdr == NULL (3) cyctim == 0, (0x7FFFFFFF - TIC_NUME)/TIC_DENO < cyctim (4) cyctim < cycphs
E_ID	Invalid ID number (only for cre_cyc) cycid<=0, VTMAX_CYC < cycid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_ccyc has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOID	No ID number available (only for acre_cyc)
E_OBJ	Object state error (only for cre_cyc) The cyclic handler indicated by cycid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_cyc creates a cyclic handler with cyclic handler ID indicated by cycid according to the content of pk_ccyc. The acre_cyc creates a cyclic handler according to the content of pk_ccyc, and returns the created cyclic handler ID.

(1) Cyclic Handler ID (cycid)

The cre_cyc creates a cyclic handler with the cyclic handler ID indicated by cycid.

(2) Cyclic Handler Attribute (cycatr)

The following are specified for cycatr.

```
cycatr:= ( TA_HLNG | [TA_STA] | [TA_PHS] )
```

- TA_HLNG (0x0000)
  Only C-language is supported for task description language.
- TA_STA (0x0002)
  The cyclic handler is in operational state.
- TA_PHS (0x0004)
  When TA_PHS is specified, the next activation time is determined preserving the activation phase when the cyclic handler is moved to operational state. When TA_PHS is not specified, the cyclic handler is activated cyctim milliseconds after issuing sta_cyc or ista_cyc.

(3) Extended Information (exinf)

When the cyclic handler is activated, exinf is passed to the cyclic handler as argument. The exinf can be widely used by the user, for example, to set information concerning the cyclic handler.

(4) Cyclic Handler Start Address (cychdr)

Specify the cyclic handler start address.

(5) Activation Cycle (cyctim), Activation Phase (cycphs)

Specify activation cycle for cyctim.

Specify activation phase from issuing this service call for cycphs. The cycphs is effective only when TA_STA or TA_PHS is specified.

5.17.2 Deletes Cyclic Handler (del_cyc)

❑C Language API

```
ER        del_cyc(ID cycid);
```

❑Parameter

cycid	Cyclic handler ID

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number cycid<=0, VTMAX_CYC < cycid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The cyclic handler indicated by cycid does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the cyclic handler indicated by cycid.

5.17.3 Starts Cyclic Handler Operation (sta_cyc, ista_cyc)

❑C Language API

```
ER          sta_cyc(ID cycid);
ER          ista_cyc(ID cycid);
```

❑Parameter

```
cycid       Cyclic handler ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID                Invalid ID number
                    cycid<=0, VTMAX_CYC < cycid
E_CTX               Context error (invoked from system state not permitted)
                      Note: The E_CTX is not detected in the following cases.
                          (1) Invocation of sta_cyc from non-task context
                          (2) Invocation of iref_cyc from task context
E_NOEXS             Non-existent object
                    The cyclic handler indicated by cycid does not exist.
```

❑Function

Places the cyclic handler indicated by cycid into an operating state.

Unless TA_PHS is specified for the cyclic handler attribute, the handler is activated each time the activation cycle elapses beginning from the time of day at which this service call was invoked.

If a cyclic handler currently in an operating state and that has had TA_PHS unspecified is specified, the service call only resets the time of day at which the cyclic handler will be activated next.

If TA_PHS is specified, the cyclic handler is activated cyclically beginning from the time of day at which it was created, so that the service call does not set the activation time.

5.17.4 Stops Cyclic Handler Operation (stp_cyc, istp_cyc)

❑C Language API

```
ER        stp_cyc(ID cycid);
ER        istp_cyc(ID cycid);
```

❑Parameter

```
cycid     Cyclic handler ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID              Invalid ID number
                  cycid<=0, VTMAX_CYC < cycid
E_CTX             Context error (invoked from system state not permitted)
                    Note: The E_CTX is not detected in the following cases.
                        (1) Invocation of stp_cyc from non-task context
                        (2) Invocation of istp_cyc from task context
E_NOEXS           Non-existent object
                  The cyclic handler indicated by cycid does not exist.
```

❑Function

Places the cyclic hander indicated by cycid into an inactive state.

5.17.5 Refers to Cyclic Handler State (ref_cyc, iref_cyc)

❑C Language API

```
ER        ref_cyc(ID cycid, T_RCYC *pk_rcyc);
ER        iref_cyc(ID cycid, T_RCYC *pk_rcyc);
```

❑Parameter

cycid	Cyclic handler ID
pk_rcyc	Pointer to the storage to which the cyclic handler state is returned

❑Packet Structure

```
typedef   struct t_rcyc {
          STAT    cycstat;    Cyclic handler operation state
          RELTIM  lefttim;    Time left before the next activation
} T_RCYC;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error pk_rcyc == NULL
E_ID	Invalid ID number cycid<=0, VTMAX_CYC < cycid
E_CTX	Context error (invoked from system state not permitted) Note: The E_CTX is not detected in the following cases. (1) Invocation of ref_cyc from non-task context (2) Invocation of iref_cyc from task context
E_MACV	Memory access violation (only for ref_cyc) The operand write access to the area indicated by pk_rcyc has not been permitted to the invoking task.
E_NOEXS	Non-existent object The cyclic handler indicated by cycid does not exist.

❑Function

Refers to the state of the cyclic hander indicated by cycid and returns its operating state (cycstat) and the remaining time before the handler is activated (lefttim) to the area pointed to by pk_rcyc.

The value returned to cycstat represents the operating state of the target cyclic handler.

- TCYC_STP (0x00000000) : Cyclic handler not operating
- TCYC_STA (0x00000001) : Cyclic handler operating

The value returned to lefttim represents a remaining time relative to the time of day at which the target cyclic handler will be activated next. If the handler will activate at the next time-tick, 0 is returned.

If the target cyclic handler is not operating, the value of lefttim is indeterminate.

5.18 Time Management Function (Alarm Handler)

Table 5.32 shows specifications of the alarm handler function.

Table 5.32 Specifications of the Alarm Handler Function

No.	Item	Content
1	Alarm handler ID	1 - VTMAX_ALM *1
2	Activation time	0 - (0x7FFFFFFF - TIC_NUME)/TIC_DENO *2
3	Extension information (parameters passed to handler)	32 bits
4	Alarm handler attribute	TA_HLNG : Written in high-level language

Notes: 1 This is the macro output to kernel_id.h by cfg600px, which indicates the maximum alarm handler ID.

2 TIC_NUME and TIC_DENO are the macros output to kernel_id.h by cfg600px, which respectively denote the numerator (system.tic_nume) and the denominator (system.tic_deno) of the time tick cycle defined in the cfg file.

Table 5.33 Service Calls for Alarm Handler Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	cre_cyc		Creates alarm handler	T		E	D	U	
2	acre_cyc		Creates alarm handler (automatic ID assignment)	T		E	D	U	
3	del_cyc		Deletes alarm handler	T		E	D	U	
4	sta_alm		Starts alarm handler operation	T		E	D	U	
5	ista_alm				N	E	D	U	
6	stp_alm		Stops alarm handler operation	T		E	D	U	
7	istp_alm				N	E	D	U	
8	ref_alm		Refers to alarm handler state	T		E	D	U	
9	iref_alm				N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.18.1 Creates Alarm Handler (cre_alm, acre_alm)

❑C Language API

```
ER        cre_alm(ID almid, T_CALM *pk_calm);
ER_ID     acre_alm(T_CALM *pk_calm);
```

❑Parameter

almid	Alarm handler ID
pk_calm	Pointer to the packet containing the alarm handler creation information

❑Packet Structure

```
typedef   struct t_calm {
          ATR     almatr;          Alarm handler attribute
          VP_INT  exinf;           Extended information
          FP      almhdr;          Alarm handler start address
} T_CALM;
```

❑Return Value

In cre_alm : E_OK for normal completion or error code
In acre_alm : Created alarm handler ID (a positive value) or error code

❑Error Code

E_RSATR	Reserved attribute almatr != TA_HLNG
E_PAR	Parameter error (1) pk_calm == NULL (2) almhdr == NULL
E_ID	Invalid ID number (only for cre_alm) almid<=0, VTMAX_ALM < almid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_calm has not been permitted to the invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOID	No ID number available (only for acre_alm)
E_OBJ	Object state error (only for cre_alm) The alarm handler indicated by almid exists.

❑Function

This service call can be called from the task that belong to trusted domain.

The cre_alm creates a alarm handler with alarm handler ID indicated by almid according to the content of pk_calm. The acre_alm creates a alarm handler according to the content of pk_calm, and returns the created alarm handler ID.

(1) Alarm Handler ID (almid)

The cre_alm creates a alarm handler with the alarm handler ID indicated by almid.

(2) Alarm Handler Attribute (almatr)

Only TA_HLNG can be specified for almatr.

- TA_HLNG (0x0000)
  Only C-language is supported for alarm handler description language.

(3) Extended Information (exinf)

When the alarm handler is activated, exinf is passed to the alarm handler as argument. The exinf can be widely used by the user, for example, to set information concerning the alarm handler.

(4) Alarm Handler Start Address (almhdr)

Specify the alarm handler start address.

5.18.2 Deletes Alarm Handler (del_alm)

❑C Language API

```
ER        del_alm(ID almid);
```

❑Parameter

```
almid     Alarm handler ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID             Invalid ID number
                 almid<=0, VTMAX_ALM < almid
E_CTX            Context error (invoked from system state not permitted)
E_MACV           Memory access violation
                 Stack pointer points out of user stack for invoking task.
E_OACV           Object access violation
                 The invoking task does not belong to trusted domain.
E_NOEXS          Non-existent object
                 The alarm handler indicated by almid does not exist.
```

❑Function

This service call can be called from the task that belong to trusted domain.

This service call deletes the alarm handler indicated by almid.

5.18.3 Starts Alarm Handler Operation (sta_alm, ista_alm)

❑C Language API

```
ER        sta_alm(ID almid, RELTIM almtim);
ER        ista_alm(ID almid, RELTIM almtim);
```

❑Parameter

```
almid     Alarm handler ID
almtim    Activation time
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

```
E_PAR             Parameter error
                  (0x7FFFFFFF - TIC_NUME)/TIC_DENO < almtim
E_ID              Invalid ID number
                  almid<=0, VTMAX_ALM < almid
E_CTX             Context error (invoked from system state not permitted)
                    Note: The E_CTX is not detected in the following cases.
                        (1) Invocation of sta_alm from non-task context
                        (2) Invocation of ista_alm from task context
E_NOEXS           Non-existent object
                  The alarm handler indicated by almid does not exist.
```

❑Function

Sets the activation time of the alarm handler indicated by almid so that the handler will start operating a relative time (specified by almtim) after the time of day at which the service call was invoked.

If an alarm handler already operating is specified, the service call clears the previously set activation time and sets a new activation time.

If the value 0 is specified for almtim, the alarm handler is activated at the next time tick.

5.18.4 Stops Alarm Handler (stp_alm, istp_alm)

❑C Language API

```
ER        stp_alm(ID almid);
ER        istp_alm(ID almid);
```

❑Parameter

```
almid     Alarm handler ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID              Invalid ID number
                  almid<=0, VTMAX_ALM < almid
E_CTX             Context error (invoked from system state not permitted)
                    Note: The E_CTX is not detected in the following cases.
                        (1) Invocation of stp_alm from non-task context
                        (2) Invocation of istp_alm from task context
E_NOEXS           Non-existent object
                  The alarm handler indicated by almid does not exist.
```

❑Function

Clears the activation time that is set for the alarm handler indicated by almid, thereby causing the handler to stop operating.

5.18.5 Refers to Alarm Handler State (ref_alm, iref_alm)

❑C Language API

```
ER        ref_alm(ID almid, T_RALM *pk_ralm);
ER        iref_alm(ID almid, T_RALM *pk_ralm);
```

❑Parameter

almid	Alarm handler ID
pk_ralm	Pointer to the storage to which the alarm handler state is returned

❑Packet Structure

```
typedef    struct t_ralm {
           STAT    almstat;     Alarm handler operation state
           RELTIM  lefttim;     Time left before the activation
} T_RALM;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

```
E_PAR              Parameter error
                   pk_ralm == NULL
E_ID               Invalid ID number
                   almid<=0, VTMAX_ALM < almid
E_CTX              Context error (invoked from system state not permitted)
                     Note: The E_CTX is not detected in the following cases.
                         (1) Invocation of ref_alm from non-task context
                         (2) Invocation of iref_alm from task context
E_MACV             Memory access violation (only for ref_alm)
                   The operand write access to the area indicated by pk_ralm has not been
                   permitted to the invoking task.
E_NOEXS            Non-existent object
                   The alarm handler indicated by almid does not exist.
```

❑Function

Refers to the state of the alarm hander indicated by almid and returns its operating state (almstat) and the remaining time before the handler is activated (lefttim) to the area pointed to by pk_ralm.

The value returned to almstat represents the operating state of the target alarm handler.

- TALM_STP (0x00000000) : Alarm handler not operating
- TALM_STA (0x00000001) : Alarm handler operating

The value returned to lefttim represents a relative remaining time before the target alarm handler will be activated. If the handler will activate at the next time-tick, 0 is returned.

If the target alarm handler is not operating, the value of lefttim is indeterminate.

5.19 System State Management Function

Table 5.34 Service Calls for System State Management Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	rot_rdq	[S][B]	Rotates task precedence	T		E	D	U	
2	irot_rdq	[S][B]			N	E	D	U	
3	get_tid	[S][B]	Refers to task ID in the RUNNING state	T		E	D	U	
4	iget_tid	[S]			N	E	D	U	
5	loc_cpu	[S][B]	Locks the CPU	T		E	D	U	L
6	iloc_cpu	[S]			N	E	D	U	L
7	unl_cpu	[S][B]	Unlocks the CPU	T		E	D	U	L
8	iunl_cpu	[S]			N	E	D	U	L
9	dis_dsp	[S][B]	Disable dispatching	T		E	D	U	
10	ena_dsp	[S][B]	Enable dispatching	T		E	D	U	
11	sns_ctx	[S]	Refers to contest state	T	N	E	D	U	L
12	sns_loc	[S]	Refers to CPU-locked state	T	N	E	D	U	L
13	sns_dsp	[S]	Refers to dispatching-disabled state	T	N	E	D	U	L
14	sns_dpn	[S]	Refers to dispatching-pending state	T	N	E	D	U	L
15	vsta_knl	[V]	Starts kernel	T	N	E	D	U	L
16	ivsta_knl	[V]		T	N	E	D	U	L
17	vsys_dwn	[V]	Terminates system	T	N	E	D	U	L
18	ivsys_dwn	[V]		T	N	E	D	U	L

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.19.1 Rotates Task Precedence (rot_rdq, irot_rdq)

❑C Language API

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

❑Parameter

```
tskpri    Task priority
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR             Parameter error
                  (1) tskpri < 0, TMAX_MPRI< tskpri
                  (2) In an invocations from non-task context, tskid == 0
E_CTX             Context error (invoked from system state not permitted)
E_MACV            Memory access violation (only for rot_rdq)
                  Stack pointer points out of user stack for invoking task.
```

❑Function

Rotates the ready queue with its priority indicated by tskpri by removing the task tied at the top of the queue and then tying it anew to the tail end of the queue.

If tskpri = TPRI_SELF (= 0) is specified for rot_rdq, the ready queue that has the invoking task's base priority is rotated.

Note that although the base priority of a task is the same as its current priority unless the message buffer function is used, if the task has a message buffer being locked to it, its base priority and current priority generally do not match. Therefore, while a message buffer is locked, even if TPRI_SELF is specified for rot_rdq, it is impossible to rotate the ready queue whose priority matches that of the invoking task.

5.19.2 Refers to Task ID in the RUNNING State (get_tid, iget_tid)

❑C Language API

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

❑Parameter

```
p_tskid   Pointer to the storage to which the task ID is returned
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR             Parameter error
                  p_tskid == NULL
E_CTX             Context error (invoked from system state not permitted)
                    Note: The E_CTX is not detected in the following cases.
                        (1) Invocation of get_tid from non-task context
                        (2) Invocation of iget_tid from task context
E_MACV            Memory access violation (only for get_tid)
                  The operand write access to the area indicated by p_tskid has not been
                  permitted to the invoking task.
```

❑Function

Returns the task ID currently in the RUNNING state to the area pointed to by p_tskid.

Specifically, if the service call is invoked from a task context, the ID of the invoking task is returned; if invoked from a non-task context, the ID of the task that was being executed then is returned. If tasks in an execution state are nonexistent, TSK_NONE (= 0) is returned.

5.19.3 Locks the CPU (loc_cpu, iloc_cpu)

❑C Language API

```
ER        loc_cpu(void);
ER        iloc_cpu(void);
```

❑Parameter

None

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted) Note: The E_CTX is not detected in the following cases. (1) Invocation of loc_cpu from non-task context (2) Invocation of iloc_cpu from task context
E_ILUSE	Illegal use of service call loc_cpu is called while the interrupt mask has been changed to other than 0 by using chg_ims.

❑Function

Places the system state into the CPU-locked state.

Characteristics of the CPU-locked state are outlined below.

(1) While in the CPU-locked state, task scheduling is not performed. (refer to supplement)

(2) The interrupts whose priority levels are lower than or equal to system.system_IPL (kernel interrupt mask level) specified in the cfg file are masked. (The IPL bits of the PSW register are changed to system.system_IPL.)

(3) Only the following service calls are invocable from the CPU-locked state:

- ext_tsk
- exd_tsk
- loc_cpu, iloc_cpu
- unl_cpu, iunl_cpu
- sns_ctx
- sns_loc
- sns_dsp
- sns_dpn
- vsys_dwn, ivsys_dwn

The system is freed from the CPU-locked state by one of the following operations:

(a) Invocation of the unl_cpu or iunl_cpu service call

(b) Invocation of the ext_tsk service call (including return from a task entry function)

(c) Invocation of the exd_tsk service call

Transition between the CPU-locked state and the CPU-unlocked state occurs only when the loc_cpu, iloc_cpu, unl_cpu, iunl_cpu, or ext_tsk service call is executed. It is necessary that when any kernel-interrupt handler or time event handler is finished, the system must be in the CPU-unlocked state. If not, the system will go down at ret_int service call. Note that when any of these handlers starts, the system is always in the CPU-unlocked state.

When the loc_cpu is called while the interrupt mask has been changed to other than 0 by using chg_ims, loc_cpu returns E_ILUSE error.

Even if this service call is invoked again while the system is already in the CPU-locked state, no errors are assumed, but the request is not queued.

❑Supplement

The CPU-locked state and the dispatching-disabled state are managed independently of each other.

5.19.4 Unlocks the CPU (unl_cpu, iunl_cpu)

❑C Language API

```
ER        unl_cpu(void);
ER        iunl_cpu(void);
```

❑Parameter

None

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (only for unl_cpu) Stack pointer points out of user stack for invoking task.

❑Function

Frees the system from the CPU-locked state.

Concretely, the unl_cpu enables task-dispatching, and changes the interrupt mask (PSW.IPL) to 0. However, the dispatching-disabled state continues after calling the unl_cpu when the loc_cpu has been called in the dispatching-disabled state by the dis_dsp. (refer to supplement)

The iunl_cpu returns the interrupt mask to just before iunl_cpu.

It is necessary to release CPU-locked state before the handler ends when iloc_cpu is used by the handler.

Even if this service call is invoked from the CPU-unlocked state, no errors are assumed, but the request is not queued.

❑Supplement

The CPU-locked state and the dispatching-disabled state are managed independently of each other. Therefore, in the unl_cpu and iunl_cpu service calls, the dispatching-disabled state is not canceled.

5.19.5 Disables Dispatching (dis_dsp)

❑C Language API

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

❑Parameter

None

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)

❑Function

Places the system state into the dispatching-disabled state.

Characteristics of the dispatching-disabled state are outlined below.

(1) Since task scheduling is no longer performed, in no case will tasks except the invoking task be transitioned to the RUNNUNG state.
(2) Interrupts are accepted.
(3) No service calls can be invoked that will result in tasks being placed into the WAITING state.

The operation that changes in the dispatching-disabled state is as follows.

(1) Invocation of the dis_dsp service call
(2) Changes the interrupt mask (PSW.IPL) to other than 0 by using the chg_ims service call

The operation that changes in the dispatching-enabled state is as follows.

(1) Invocation of the ena_dsp service call
(2) Invocation of the ext_tsk service call (including return from a task entry function)
(3) Invocation of the exd_tsk service call
(4) Changes the interrupt mask (PSW.IPL) to 0 by using the chg_ims service call

Be aware that while the system is placed in the dispatching-disabled state, even if the status of the invoking task is inspected by the ref_tsk, iref_tsk, ref_tst, or iref_tst service call, the task status may not appear to be in an execution state.

Even if this service call is invoked again while the system is already in the dispatching-disabled state, no errors are assumed, but the request is not queued.

5.19.6 Enables Dispatching (ena_dsp)

❑C Language API

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

❑Parameter

None

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.

❑Function

Frees the system from the dispatching-disabled state that was set by the dis_dsp or chg_ims service call and performs scheduling of tasks.

Even if this service call is invoked from the dispatching-enabled state, no errors are assumed, but the request is not queued.

5.19.7 Refers to Context State (sns_ctx)

❑C Language API

```
BOOL        sns_ctx(void);
```

❑Parameter

None

❑Return Value

TRUE or FALSE or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)

❑Function

This service call returns TRUE when the current context type is "non-task context" and return FALSE when the current context type is "task context".

This service call can also be invoked from the CPU-locked state.

5.19.8 Refers to CPU-Locked State (sns_loc)

❑C Language API

```
BOOL        sns_loc(void);
```

❑Parameter

None

❑Return Value

TRUE or FALSE or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)

❑Function

This service call returns TRUE when the system is in the CPU-locked state and returns FALSE when the system state is in the CPU-unlocked state.

This service call can also be invoked from the CPU-locked state.

5.19.9 Refers to Dispatching-Disabled Dtate (sns_dsp)

❑C Language API

```
BOOL        sns_dsp(void);
```

❑Parameter

None

❑Return Value

TRUE or FALSE or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)

❑Function

This service call returns TRUE when the system is in the dispatching disabled state and returns FALSE when the system state is in the dispatching enabled state.

This service call can also be invoked from the CPU-locked state.

5.19.10 Refers to Dispatching-Pending State (sns_dpn)

❑C Language API

```
BOOL        sns_dpn(void);
```

❑Parameter

None

❑Return Value

TRUE or FALSE or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)

❑Function

This service call returns TRUE when the system is in the dispatching pending state and returns FALSE when the system state is in any other state.

The case where either the following is filled is called dispatching pending state.

(1) The system is in the dispatching-disabled state.
(2) The system is in the CPU-locked state.
(3) The application is being executed in the non-task context.

This service call can also be invoked from the CPU-locked state.

5.19.11 Starts Kernel (vsta_knl, ivsta_knl)

❑C Language API

```
void      vsta_knl(void);
void      ivsta_knl(void);
```

❑Parameter

None

❑Function

Activates the kernel. In no case does the application return from this service call.

The following outlines processing performed by these service calls.

1. Sets MPU (Memory Protection Unit) to all access inhibits.
2. Initializes INTB register to the start address of the relocatable interrupt vector table generated by cfg600px.
3. Initializes kernel internal tables.
4. Creates objects defined by cfg file.
5. Initializes system timer (call _RI_init_cmt_knl())
   Refer to "7.2(2)Timer Initialization Call-back Function (_RI_init_cmt_knl())".
6. Enters the multitasking environment

When an error is detected in the above processing, system goes down.

This service call must be called in the following states.

1. The CPU must not accept any interrupt. (ex. PSW.I=0)
2. Supervisor mode (PSW.PM=0)

Note that E_CTX errors are not detected in this service call.

This service call can be called when the interrupt mask level (PSW.IPL) is higher than the kernel interrupt mask level (system.system_IPL)

This service call is the function outside the range of µITRON 4.0 specifications

5.19.12 Terminates System (vsys_dwn, ivsys_dwn)

❑C Language API

```
void vsys_dwn(W type, VW inf1, VW inf2, VW inf3);
void ivsys_dwn(W type, VW inf1, VW inf2, VW inf3);
```

❑Parameter

```
type      Error type
inf1      System abnormal information 1
inf2      System abnormal information 2
inf3      System abnormal information 3
```

❑Function

Passes control to the system-down routine.

For type, specify a value (1 - 0x7FFFFFFF) that represents the type of error that occurs. Note that the values equal to or less than 0 are reserved for use by the system.

The system-down routine is also invoked when an abnormal condition in the kernel is detected.

This service call can be invoked from any state.

In no case will the application return from this service call.

Furthermore, E_CTX errors are not detected in this service call.

For details about parameter specifications, see Section 6.8, "System-Down Routine."

This service call can be called when the interrupt mask level (PSW.IPL) is higher than the kernel interrupt mask level (system.system_IPL)

This service call is the function outside the range of µITRON 4.0 specifications.

5.20 Interrupt management Function

Table 5.35 Service Calls for Interrupt Management Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	chg_ims		Changes interrupt mask	T		E	D	U	
2	ichg_ims				N	E	D	U	
3	get_ims		Refers to interrupt mask	T		E	D	U	
4	iget_ims				N	E	D	U	
5	ret_int	[S][B]	Returns from kernel interrupt handler		N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.20.1 Changes Interrupt Mask (chg_ims, ichg_ims)

❑C Language API

```
ER          chg_ims(IMASK imask);
ER          ichg_ims(IMASK imask);
```

❑Parameter

```
imask       Interrupt mask
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR             Parameter error
                  imask > 15
E_CTX             Context error (invoked from system state not permitted)
E_MACV            Memory access violation (only for chg_ims)
                  Stack pointer points out of user stack for invoking task.
```

❑Function

Changes the CPU's interrupt mask (PSW.IPL) to the value specified by imask.

For imask, any value in the range 0–15 can be specified.

In the chg_ims service call, the system state shifts to the dispatching-disabled state when specified imask is other than 0 (this is equivalent to the dis_dsp service call.) And the system state shifts to the dispatching-enabled state when specified imask is 0 (this is equivalent to the ena_dsp service call.) On the other hand, the ichg_ims service call doesn't cause the transition of the dispatching-disabled state and the dispatching-enabled state.

Note, the PSW register is handled as the task context register. (refer to Supplement #2)

This service call can be called when the interrupt mask level (PSW.IPL) is higher than the kernel interrupt mask level (system.system_IPL).

It is necessary to return it based on the interrupt mask before task or handler is ended.

❑Supplement

1. In the non-task context, the interrupt mask must not be lowered more than the value at the initiation.

2. The system state shifts to the dispatching-enabled state at that time if the ena_dsp is called after the interrupt mask (PSW.IPL) is changed other than 0.
   The interrupt mask is changed in the state of the task of the dispatch destination when dispatching it to another task because the PSW is handled as the task context register.

3. The loc_cpu returns E_ILUSE error while having changed the interruption mask excluding 0.

4. When the interrupt mask is higher than the kernel interrupt mask level (system.system_IPL), service calls that can use it is limited as follows.
   chg_ims, ichg_ims, get_ims, iget_ims, vsta_knl, ivsta_knl, vsys_dwn, ivsys_dwn

5.20.2 Refers to Interrupt Mask (get_ims, iget_ims)

❑C Language API

```
ER        get_ims(IMASK *p_imask);
ER        iget_ims(IMASK *p_imask);
```

❑Parameter

p_imask   Pointer to storage to which the interrupt mask level is returned

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error p_imask == NILL
E_MACV	Memory access violation (only for get_ims) The operand write access to the area indicated by p_imask has not been permitted to the invoking task.

❑Function

Returns the current interrupt mask level (PSW.IPL) to the area pointed to by p_imask.

Note that E_CTX errors are not detected in this service call.

This service call can be called when the interrupt mask level (PSW.IPL) is higher than the kernel interrupt mask level (system.system_IPL)

5.20.3 Returns from kernel interrupt handler (ret_int)

❑C Language API

```
None (This service call is invoked automatically at termination of an interrupt handler according
to its definition in the cfg file.)
Service calls ret_int does not return to the position where it was issued.
When the following error is detected, the system will go down.
E_CTX            Context error (invoked from system state not permitted)
E_MACV           Memory access violation
                 Stack pointer points out of user stack for interrupted task.
```

❑Function

Performs processing for return from a kernel interrupt handler.

When returning from a kernel interrupt handler which occurred while executing in task-context, the scheduler is put to work and tasks are switched depending on the result.

Other case, returns to the interrupted program.

The interrupt mask level (PSW.IPL) at the time a kernel interrupt hander is finished, i.e., when this service call is invoked must be less than or equal to the kernel interrupt mask (system.system_IPL). Otherwise, a context error is detected in this service call and system-down results. The following cases apply to this:

(1) Cases where a non-kernel interrupt handler is erroneously defined as a kernel interrupt handler
(2) Cases where PSW.IPL is changed to less than the kernel mask level (system.system_IPL) in a kernel interrupt handler and the changed mask level is left intact when the handler is finished

In addition, when this service call is issued in the CPU-locked state or from task-context, a system-down results.

❑Supplement

There are two types of interrupts: one kernel interrupt and one non-kernel interrupt. For details, see Section 2.7.1, "Type of Interrupt."

To define a kernel interrupt handler, specify YES for interrupt_vector[].os_int in the cfg file. In this case, when the relevant handler function is compiled, an object code is generated that invokes this service call at the end of the handler.

On the other hand, the non-kernel interrupt handler is defined differently depending on whether it is a relocatable vector or a fixed vector interrupt. To define the former, specify NO for interrupt_vector[].os_int in the cfg file; to define the latter, use interrupt_fvector[]. In these cases, when the relevant handler function is compiled, an object code is generated that causes the handler to be returned to the main at the end of it by an RTE instruction.

5.21 System Configuration Management Function

Table 5.36 Service Calls for System Configuration Management Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	ref_ver		Refers to version information	T		E	D	U	
2	iref_ver		Refers to version information		N	E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.21.1 Refers to Version Information (ref_ver, iref_ver)

❑C Language API

```
ER          ref_ver(T_RVER *pk_rver);
ER          iref_ver(T_RVER *pk_rver);
```

❑Parameter

pk_rver   Pointer to storage to which the version formation is returned

❑Packet Structure

```
typedef    struct {
           UH      maker;       Kernel maker's code
           UH      prid;        Identification number of the kernel
           UH      spver;       Version number of the ITRON specification
           UH      prver;       Version number of the kernel
           UH      prno[4];     Management information of the kernel product
} T_RVER;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error pk_rver == NULL
E_CTX	Context error (invoked from system state not permitted) Note: The E_CTX is not detected in the following cases. (1) Invocation of ref_ver from non-task context (2) Invocation of iref_ver from task context
E_MACV	Memory access violation (only for ref_ver) The operand write access to the area indicated by pk_rver has not been permitted to the invoking task.

❑Function

Reads out information about the kernel version currently being executed and returns the result to the area pointed to by pk_rver.

The packet pointed to by pk_rver will have the following information returned to it:

(1) Kernel Maker's Code (maker)

Represents the manufacturer who created the kernel. This kernel returns 0x011B that is the maker code assigned to Renesas Electronics Corporation as maker.

(2) Identification Number of the Kernel (prid)

Represents the number that identifies the kernel and the type of VLSI. This kernel returns 0x0004 as prid.

(3) Version Number of the ITRON Specification (spver)

Represents the specification to which the kernel conforms. This information consists of separate bit fields, each having the following meaning:

- bit15 - 12 : MAGIC (number to identify a series in TRON specifications)
  This kernel returns 0x5. (μTRON specification)
- bit 11 - 0 : SpecVer (version number of TRON specification on which product is based)
  This kernel returns 0x403. (μTRON 4.0 specification Ver.4.03)

(4) Version Number of the Kernel (prver)

Represents the version number of the kernel.

The value of prver differs with each product version. For example, prver for V.1.00 Release 00 is 0x0100.

(5) Management Information of the Kernel Product (prno)

Represents product management information and product number, etc.

The value from prno[0] to prno[3] for the kernel described herein is 0x0000.

5.22 Object Reset Function

The object reset function is the function to revert various objects to their initial state. This function is outside µITRON 4.0 specification.

Table 5.37 Service Calls for Object Reset Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	vrst_dtq	[V]	Resets data queue	T		E	D	U	
2	vrst_mbx	[V]	Resets mailbox	T		E	D	U	
3	vrst_mbf	[V]	Resets message buffer	T		E	D	U	
4	vrst_mpf	[V]	Resets fixed-sized memory pool	T		E	D	U	
5	vrst_mpl	[V]	Resets variable-sized memory pool	T		E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings:
"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.22.1 Resets Data Queue (vrst_dtq)

❑C Language API

```
ER        vrst_dtq( ID dtqid );
```

❑Parameter

```
dtqid     Data queue ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID            Invalid ID number
                dtqid<=0, VTMAX_DTQ < dtqid
E_CTX           Context error (invoked from system state not permitted)
E_MACV          Memory access violation
                Stack pointer points out of user stack for invoking task.
E_NOEXS         Non-existent object
                The data queue indicated by dtqid does not exist.
```

❑Function

Resets the data queue indicated by dtqid.

More specifically, this service call clears the data stored in the data queue and, if there is any task waiting to send to the data queue, frees those tasks from the WAITING state. In this case, the task freed from the WAITING state will have error code EV_RST returned to it.

Note that tasks waiting to receive from the data queue are not freed from the WAITING state.

This service call is the function outside µITRON 4.0 specification.

5.22.2 Resets Mailbox(vrst_mbx)

❑C Language API

```
ER          vrst_mbx( ID mbxid );
```

❑Parameter

mbxid	Mailbox ID

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number mbxid<=0, VTMAX_MBX < mbxid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_NOEXS	Non-existent object The mailbox indicated by mbxid does not exist.

❑Function

Resets the mailbox indicated by mbxid.

More specifically, this service call empties the message queue.

This service call is the function outside µITRON 4.0 specification.

5.22.3 Resets Message Buffer (vrst_mbf)

❑C Language API

```
ER        vrst_mbf( ID mbfid );
```

❑Parameter

```
mbfid     Message buffer ID
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_ID              Invalid ID number
                  mbfid<=0, VTMAX_MBF < mbfid
E_CTX             Context error (invoked from system state not permitted)
E_MACV            Memory access violation
                  Stack pointer points out of user stack for invoking task.
E_NOEXS           Non-existent object
                  The message buffer indicated by mbfid does not exist.
```

❑Function

Resets the message buffer indicated by mbfid.

More specifically, this service call clears the messages stored in the message buffer and, if there is any task waiting to send to the message buffer, frees those tasks from the WAITING state. In this case, the task freed from the WAITING state will have error code EV_RST returned to it.

Note that tasks waiting to receive from the message buffer are not freed from the WAITING state.

This service call is the function outside μITRON 4.0 specification.

5.22.4 Resets Fixed-sized Memory Pool (vrst_mpf)

❑C Language API

```
ER         vrst_mpf( ID mpfid );
```

❑Parameter

mpfid	Fixed-sized memory pool ID

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number mpfid<=0, VTMAX_MPF < mpfid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_NOEXS	Non-existent object The fixed-sized memory pool indicated by mpfid does not exist.

❑Function

Resets the fixed-sized memory pool indicated by mpfid.

More specifically, this service call changes the status of all memory blocks to an unused state and, if there is any task waiting to get a memory block, frees those tasks from the WAITING state. In this case, the task freed from the WAITING state will have error code EV_RST returned to it.

Since all memory blocks are handled as in 'unused' state, once this service call is executed, the memory blocks previously acquired, if any, cannot be used.

This service call is the function outside μITRON 4.0 specification.

5.22.5 Resets Variable-sized Memory Pool (vrst_mpl)

❑C Language API

```
ER        vrst_mpl( ID mplid );
```

❑Parameter

```
mplid     Variable-sized memory pool ID
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_ID	Invalid ID number mplid<=0, VTMAX_MPL < mplid
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_NOEXS	Non-existent object The variable-sized memory pool indicated by mplid does not exist.

❑Function

Resets the variable-sized memory pool indicated by mplid.

More specifically, this service call changes the status of all memory blocks in the variable-sized memory pool to an unused state and, if there is any task waiting to get a memory block, frees those tasks from the WAITING state. In this case, the task freed from the WAITING state will have error code EV_RST returned to it.

Since all memory blocks are handled as in 'unused' state, once this service call is executed, the memory blocks previously acquired, if any, cannot be used.

This service call is the function outside µITRON 4.0 specification.

5.23 Memory Object Management Function

Table 5.38 Service Calls for Memory Object Management Function

No.	Service Call *1		Description	System State *2					
				T	N	E	D	U	L
1	ata_mem		Registers memory object	T		E	D	U	
2	det_mem		Unregisters memory object	T		E	D	U	
3	sac_mem		Changes access permission vector for memory object	T		E	D	U	
4	vprb_mem	[V]	Checks access permission	T		E	D	U	
5	ref_mem		Refers to memory object state	T		E	D	U	

Notes: 1 The symbol "[S]" denotes µITRON 4.0-compliant, standard-profile service calls; "[B]" denotes µITRON 4.0-compliant, basic-profile service calls; and "[V]" denotes service calls other than µITRON 4.0 specification.

2 The letters representing the system state have the following meanings :

"T" invocable from task contexts, "N" invocable from non-task contexts, "E" invocable from a dispatching-enabled state, "D" invocable from a dispatching-disabled state, "U" invocable from the CPU-unlocked state, "L" invocable from the CPU-locked state.

5.23.1 Registers Memory Object (ata_mem)

❑C Language API

```
ER          ata_mem(T_AMEM *pk_amem, ACVCT *p_acvct);
```

❑Parameter

pk_amem	Pointer to the packet containing the memory object registration information
p_acvct	Pointer to the access permission vector

❑Packet Structure

```
typedef     struct t_amem {
            ATR     382matr;            Memory object attribute
            VP      base;               memory object start address
            SIZE    size;               Size of memory object (in bytes)
} T_AMEM;

typedef struct acvct {
            ACPTN   acptn1;             Access permission pattern for operand read
            ACPTN   acptn2;             Access permission pattern for operand write
            ACPTN   acptn3;             Access permission pattern for execution
} ACVCT;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error (1) pk_amem == NULL (2) base is not 16-bytes boundary. (3) size is not multiple of 16. (4) p_acvct == NULL (5) acptn1 == acptn2 == acptn3 == 0 (6) Bits corresponded to the domain ID that is larger than the maximum domain ID of either acptn1, acptn2 and acptn3 is 1. (7) base + size > 0x10000000 (8) size == 0
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by pk_amem has not been permitted to the invoking task. (3) The operand read access to the area indicated by p_acvct has not been permitted to the invoking task.
E_OACV	Object access violation (1) The invoking task does not belong to trusted domain. (2) The number of memory objects from which the access is permitted to same domain exceeds 7.
E_OBJ	Object state error The memory object started from base has already registered.

❑Function

This service call can be called from the task that belong to trusted domain.

The ata_mem service call registers the area started from the address specified by base with the size [bytes] as the memory object with the access permission vector specified by p_acvct.

The memory object area must satisfy the following.

1. The start address must be 16-bytes boundary. If not, error E_PAR is returned.
2. The size must be multiple of 16. If not, error E_PAR is returned.
3. The memory object area must not either with all user stacks and all other memory objects. If not, an error is not detected and correct system operation cannot be guaranteed.

The mematr is only disregarded.

5.23.2 Unregisters Memory Object (det_mem)

❑C Language API

```
ER        det_mem(VP base);
```

❑Parameter

base	Memory object start address

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation Stack pointer points out of user stack for invoking task.
E_OACV	Object access violation The invoking task does not belong to trusted domain.
E_NOEXS	Non-existent object The memory object started from by base does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

This service call unregisters the memory object started from base.

5.23.3 Changes Access Permission Vector for Memory Object (sac_mem)

❑C Language API

```
ER        sac_mem(VP base, ACVCT *p_acvct);
```

❑Parameter

```
base      Memory object start address
p_acvct   Pointer to the access permission vector
```

❑Packet Structure

```
typedef struct acvct {
          ACPTN   acptn1;          Access permission pattern for operand read
          ACPTN   acptn2;          Access permission pattern for operand write
          ACPTN   acptn3;          Access permission pattern for execution
} ACVCT;
```

❑Return Value

E_OK for normal completion or error code

❑Error Code

E_PAR	Parameter error (1) p_acvct == NULL (2) acptn1 == acptn2 == acptn3 == 0 (3) bits corresponded to the domain ID that is larger than the maximum domain ID of either acptn1, acptn2 and acptn3 is 1.
E_CTX	Context error (invoked from system state not permitted)
E_MACV	Memory access violation (1) Stack pointer points out of user stack for invoking task. (2) The operand read access to the area indicated by p_acvct has not been permitted to the invoking task.
E_OACV	Object access violation (1) The invoking task does not belong to trusted domain. (2) The number of memory objects from which the access is permitted to same domain exceeds 7.
E_NOEXS	Non-existent object The memory object started from by base does not exist.

❑Function

This service call can be called from the task that belong to trusted domain.

The sac_mem changes the access permission vector for the memory object started from base to the content indicated by p_acvct.

5.23.4 Checks Access Permission (vprb_mem)

❑C Language API

```
ER        vprb_mem(VP base, SIZE size, ID tskid, MODE pmmode);
```

❑Parameter

```
base      Start address for checking
size      Size of checking area (in bytes)
tskid     Task ID
pmmode    Access mode
```

❑Return Value

```
TRUE or FALSE or error code
```

❑Error Code

```
E_ID              Invalid ID number
                  tskid<0, VTMAX_TSK < tskid
E_PAR             Parameter error
                  (1) size == 0
                  (2) pmmode == 0, Either of bits except bit0, bit1 and bit2 of pmmode is 1.
E_CTX             Context error (invoked from system state not permitted)
E_MACV            Memory access violation
                  Stack pointer points out of user stack for invoking task.
E_NOEXS           Non-existent object
                  The task indicated by tskid does not exist.
```

❑Function

The vprb_mem checks whether the task indicated by tskid has the access permission indicated by pmmode for the memory area of size bytes from base. The vprb_mem returns TRUE when the access is permitted and return FALSE when the access is not permitted.

The following are specified for pmmode.

```
pmmode:= (TPM_READ | TPM_WRITE | TPM_EXEC )
```

- TPM_READ (0x0001)
  Checks whether operand read access is permitted.
- TPM_WRITE (0x0002)
  Checks whether operand write access is permitted.
- TPM_EXEC (0x0004)
  Checks whether execution access is permitted.

Specifying tskid = TSK_SELF (= 0) means that the invoking task itself is specified.

This service call is the function outside μITRON 4.0 specification.

5.23.5 Refers to Memory Object State (ref_mem)

❑C Language API

```
ER        ref_mem(VP base, T_RMEM *pk_rmem);
```

❑Parameter

```
base      Start address for checking
pk_rmem   Pointer to the storage to which the memory object state is returned
```

❑Packet Structure

```
typedef struct acvct {
          ACPTN   acptn1;          Access permission pattern for operand read
          ACPTN   acptn2;          Access permission pattern for operand write
          ACPTN   acptn3;          Access permission pattern for execution
} ACVCT;

typedef   struct t_rmem {
          ACVCT   acvct;      Access permission vector
} T_AMEM;
```

❑Return Value

```
E_OK for normal completion or error code
```

❑Error Code

```
E_PAR             Parameter error
                  pk_rmem == NULL
E_CTX             Context error (invoked from system state not permitted)
E_MACV            Memory access violation
                  (1) Stack pointer points out of user stack for invoking task.
                  (2) The operand write access to the area indicated by pk_rmem has not been
                      permitted to the invoking task.
E_NOEXS           Non-existent object
                  The memory object started from by base does not exist.
```

❑Function

Refers to the state of the memory object started from base and returns its state to the area pointed to by pk_rmem.

6. How to Write Application

6.1 Header Files

Include the following header files.

- kernel.h
  This is the header file of the kernel. kernel.h is stored in the "<installation directory>\inc600."
- kernel_id.h
  kernel_id.h is output by cfg600px. It is in this kernel_id.h that the various object names (xxx[].name), kernel configuration macros specified in the cfg file and prototype declaration of tasks and handlers are defined.
  The ID name can be used for the ID number that is passed to a service call, as shown below.

```
ercd = act_tsk(ID_TASK1);
```

6.2 Handling of Variables

There is no relationship between the storage classes of variables in C language and the program types such as tasks and handlers under kernel specifications.

Table 6.1 shows how variables in C language are handled. For example, if it is possible that a global variable will be accessed from multiple tasks at the same time, exclusive control between those tasks is needed.

Table 6.1 Handling of Variables in C Language

No.	Storage class	Handling	Storage allocation
1	Global variables	Shared variables for all programs (tasks, handlers)	Static
2	Static variables outside function	Shared variables for the functions in one and the same file	Static
3	Auto variables, register variables, and static variables in function	Variables in the relevant function	Dynamic (stack)

6.3 Task

6.3.1 Coding

Figure 6.1 shows an example of how a task entry function is coded.

```
#include "kernel.h"
#include "kernel_id.h"
#pragma task Task1              /* (1) */
void  Task1(VP_INT exinf);      /* (2) */
void  Task1(VP_INT exinf)       /* (3) */
{
    /* processing */            /* (4) */
    ext_tsk();                  /* (5) */
}
```

Figure 6.1 Example of Coding a Task Entry Function

(1) Describes "#pragma task" directive for the task entry function. This description is not necessary when the task is created in the cfg file because cfg600px outputs this directive into the kernel_id.h.
(2) Describes prototype declaration for the task entry function. This description is not necessary when the task is created in the cfg file because cfg600px outputs this declaration into the kernel_id.h.
(3) The APIs of task entry functions are as described in this manual.
Table 6.2 shows value passed by exinf.

Table 6.2 Value passed by exinf

Activation Method	Value passed by exinf
"task[].initial_start = ON" in the cfg file	Task extended information
AT_ACT attribute is specified at cre_tsk or acre_tsk	
act_tsk or iact_tsk	
sta_tsk or ista_tsk	Start code specified by sta_tsk or ista_tsk

(4) In the task, any service calls invocable from task context can be used.
(5) Call ext_tsk() at a place where the task needs to be terminated. Note that a return from the task entry function has the same effect as calling ext_tsk().

A task entry function may also be written with an infinite loop, as shown in Figure 6.2.

```
#include "kernel.h"
#include "kernel_id.h"
#pragma task Task1
void  task(VP_INT exinf);
void  task(VP_INT exinf)
{
    for(;;) {
    /* processing */
     }
}
```

Figure 6.2 Example of Coding a Task Entry Function that Loops Infinitely

6.3.2 CPU State at Start

Since a task is executed in user mode, privileged instructions cannot be used. In the assembler, there is a helpful facility (-chkpm option) that produces a warning when privileged instructions are used.

Note that when a task is activated, all interrupts in an enabled state.

Table 6.3 PSW at Start of Task

No.	Bit	Initial value	Description
1	IPL	0	All interrupts are acceptable.
2	I	1	
3	PM	1	User mode
4	U	1	USP (user stack pointer)
5	C, Z, S, O	Indeterminate	
6	Other bits	0	

If system.context includes "FPSW", the initial value of the FPSW register is 0x00000100.

6.4 Task Exception Handling Routine

6.4.1 Coding

Figure 6.3 shows an example of how a task entry function is coded.

```
#include "kernel.h"
#include "kernel_id.h"
#pragma taskexception Texrtn1                      /* (1) */
void  Texrtn1(TEXPTN texptn, VP_INT exinf);        /* (2) */
void  Texrtn1(TEXPTN texptn, VP_INT exinf)         /* (3) */
{
    /* processing */                               /* (4) */
}
```

Figure 6.3 Example of Coding a Task Exception Handling Routine Entry Function

(1) Describes "#pragma taskexceptin" directive for the task exception handling routine entry function. This description is not necessary when the task exception handling routine is defined in the cfg file because cfg600px outputs this directive into the kernel_id.h.
(2) Describes prototype declaration for the task exception handling routine entry function. This description is not necessary when the task exception handling routine is defined in the cfg file because cfg600px outputs this declaration into the kernel_id.h.
(3) The APIs of task exception handling routine entry functions are as described in this manual.
The exception code is passed by texptn, and the task extended information is passed by exinf.
(4) In a time task exception handling routine, it is possible to use the service calls invocable from task context.

6.4.2 CPU State at Start

Since a task exception handling routine is executed in user mode, privileged instructions cannot be used. In the assembler, there is a helpful facility (-chkpm option) that produces a warning when privileged instructions are used.

Table 6.4 PSW at Start of Task Exception Handling Routine

No.	Bit	Initial value	Description
1	IPL	Same as the task before staring the task exception handling routine	
2	I	1	
3	PM	1	User mode
4	U	1	USP (user stack pointer)
5	C, Z, S, O	Indeterminate	
6	Other bits	0	

If system.context includes "FPSW", the initial value of the FPSW register is 0x00000100.

6.5 Interrupt Handler

6.5.1 Coding

Figure 6.4 shows an example of how an interrupt handler entry function is coded.

```
#include "kernel.h"
#include "kernel_id.h"
void  int_handler(void)           /* (1) */
{
    /* processing */              /* (2) */
}                                 /* (3) */
```

Figure 6.4 Example of Coding an Interrupt Handler Entry Function

(1) The APIs of interrupt handler entry functions are as described in this manual. The prototype declarations of interrupt handler entry functions are output to kernel_id.h by cfg600px.

(2) In the kernel interrupt handlers, it is possible to use the service calls invocable from non-task contexts. In the non-kernel interrupt handlers, however, service calls cannot be used.

(3) When a handler entry function is compiled, if it is of the kernel interrupt type, an object code is generated that invokes the ret_int service call at the end of the handler.
For non-kernel interrupt handlers, an object code is generated that causes the handler to be returned to the main at the end of it by an RTE instruction.
For fast interrupt handlers, an object code is generated that causes the handler to be returned to the main at the end of it by an RTFI instruction.

6.5.2 CPU State at Start

A interrupt handler is executed in supervisor mode.

As for the interrupt enable state, when "E" is specified for interrupt_vector[].pragma_switch, interrupts are masked with the relevant interrupt priority level.

On the other hand, when "E" is not specified for pragma_switch, and for the fixed vector interrupt case (interrupt_fvector[]), all interrupts are masked.

Table 6.5 PSW at Start of Interrupt Handler

No.	Bit	Initial value	Description
1	IPL	• Interrupt : Relevant interrupt priority level • CPU exception : Same before exception	Do not lower IPL more than the value when the handler starts.
2	I	• "E" specified for interrupt_vector[].pragma_switch: 1 • Other than the above : 0	
3	PM	0	Supervisor mode
4	U	0	ISP (interrupt stack pointer)
5	C,Z,S,O	Indeterminate	
6	Other bits	0	

6.6 Time Event Handler (Cyclic Handler and Alarm Handler)

6.6.1 Coding

The method of coding the entry functions for cyclic and alarm handlers both are the same. Figure 6.5 shows an example of how a time event handler entry function is coded.

```
#include "kernel.h"
#include "kernel_id.h"
#pragma cychandler Cychdr1                 /* (1) */
void  Cychdr1(VP_INT exinf);               /* (2) */
void  Cychdr1(VP_INT exinf)                /* (3) */
{
    /* processing */                        /* (4) */
}                                           /* (5) */
```

Figure 6.5 Example for Coding a Time Event Handler Entry Function

(1) Describes "#pragma cychandler" directive for the cyclic handler entry function or "#pragma almhandler" directive for the alarm handler entry function. This description is not necessary when the time event handler is created in the cfg file because cfg600px outputs this directive into the kernel_id.h.
(2) Describes prototype declaration for the time event handler entry function. This description is not necessary when the time event handler is created in the cfg file because cfg600px outputs this declaration into the kernel_id.h.
(3) The APIs of time event handler entry functions are as described in this manual.
The task extended information is passed by exinf.
(4) In a time event handlers, it is possible to use the service calls invocable from non-task context.
(5) A time event handlers is started by a call of function from the system clock interrupt handler in the kernel.

6.6.2 CPU State at Start

A time event handler is executed in a context of the kernel's internal system clock interrupt handler.

A time event handler is executed in supervisor mode.

As for the interrupt enable state, interrupts are masked with the timer interrupt level (clock.IPL).

Table 6.6 PSW at Start of Time Event Handler

No.	Bit	Initial value	Description
1	IPL	Timer interrupt level	Value specified as "clock.IPL" in the cfg file Do not lower IPL more than the value when the handler starts.
2	I	1	
3	PM	0	Supervisor mode
4	U	0	ISP (interrupt stack pointer)
5	C,Z,S,O	Indeterminate	
6	Other bits	0	

6.7 Access Exception Handler

6.7.1 Summary

The access exception handler is initiated when the access that has not been permitted is generate while executing in the task-context.

The system has only one access exception handler. The user must implement the access exception handler.

6.7.2 Coding

Figure 6.6 shows an example of how the access exception handler entry function is coded.

```
#include "kernel.h"
#include "kernel_id.h"
void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr);
void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr)
{
    /* processing */
}
```

Figure 6.6 Example for Coding the Access Exception Handler Entry Function

The API of access exception handler entry functions is as described in this manual. The function names are predetermined to be "_RI_sys_access_exception".

Table 6.7 shows specification of the arguments passed to access exception handler.

In the access exception handler, it is possible to use the service calls invocable from non-task context.

Table 6.7 Arguments Passed to Access Exception Handler.

No.	Argument	Register	Description
1	pc	R1	The instruction address that causes access exception
2	psw	R2	The PSW value at access exception
3	sts	R3	Factor of access exception (= Value of MPESTS register in the MPU (Memory Protection Unit))
4	addr	R4	• Operand access error : Access address (= Value of MPDEA register in the MPU) • Execution access error : Indeterminate

6.7.3 CPU State at Start

The access exception handler is executed in supervisor mode.

Table 6.8 PSW at Start of Access Exception Handler

No.	Bit	Initial value	Description
1	IPL	Same before access exception	Do not lower IPL more than the value when the handler starts.
2	I	0	All interrupts are masked.
3	PM	0	Supervisor mode
4	U	0	ISP (interrupt stack pointer)
5	C,Z,S,O	Indeterminate	
6	Other bits	0	

6.8 System-Down Routine

6.8.1 Summary

The system-down routine is a routine that is called when the system is down.

When one of the following phenomena occurs, a system-down results:

- Undefined interrupt generated
- vsys_dwn or ivsys_dwn service call issued
- Unlinked service call issued[4]
- Context error in ext_tsk
- Context error in exd_tsk
- Context error in ret_int
- Error in vsta_knl, ivsta_knl

The system-down routine must always be created by the user.

6.8.2 Coding

Figure 6.7 shows an example of how a system-down routine entry function is coded.

```
#include "kernel.h"
#include "kernel_id.h"
void  _RI_sys_dwn__(W type, VW inf1, VW inf2, VW inf3);
void  _RI_sys_dwn__(W type, VW inf1, VW inf2, VW inf3) /
{
    /* processing */
    while(1);
}
```

Figure 6.7 Example for Coding a System-Down Routine Entry Function

The API of system-down routine entry functions is as described in this manual. The function names are predetermined to be "_RI_sys_dwn__."

No service calls can be invoked in the system-down routine.

In no case can the application return from the entry function of the system-down routine.

The specification of the arguments passed to system-down routine is shown below.

(1) type == -1 (Error in ret_int)

Table 6.9 Arguments Passed to System-Down Routine (type == -1)

inf1	inf2	inf3	Description
E_CTX	2	Indeterminate	The ret_int is called from task context
	3	Indeterminate	The ret_int is called in the state "PSW.IPL > kernel interrupt mask level".
	5	Indeterminate	The ret_int is called in the CPU-locked state.
E_MACV	12	Indeterminate	Stack pointer points out of user stack for interrupted task.

[4] Refer to 9.4, "Notes"

(2) type == -2 (Error in ext_tsk)

Table 6.10 Arguments Passed to System-Down Routine (type == -2)

inf1	inf2	inf3	Description
E_CTX	1	Indeterminate	The ext_tsk is called from non-task context.
	4	Indeterminate	The ext_tsk is called in the state "PSW.IPL > system.system_IPL".

(3) type == -3 (Error by using unlinked service call)

Table 6.11 Arguments Passed to System-Down Routine (type == -3)

inf1	inf2	inf3	Description
E_NOSPT	Indeterminate		Unlinked service call is issued.

(4) type == -4 (Error at returning from a task exception handling routine)

Table 6.12 Arguments Passed to System-Down Routine (type == -4)

inf1	inf2	inf3	Description
E_CTX	7	Indeterminate	A task exception handling routine returns in the state "PSW.IPL > system.system_IPL".
	8	Indeterminate	A task exception handling routine returns in the CPU-locked state.
	9	Indeterminate	A task exception handling routine returns in the non-task context.

(5) type == -5 (Error in exd_tsk)

Table 6.13 Arguments Passed to System-Down Routine (type == -5)

inf1	inf2	inf3	Description
E_CTX	10	Indeterminate	The exd_tsk is called in the state "PSW.IPL > system.system_IPL".
	11	Indeterminate	The exd_tsk is called from non-task context.

(6) type == -6 (Error in vsta_knl and ivsta_knl)

Table 6.14 Arguments Passed to System-Down Routine (type == -6)

<table>
<tr><th>inf1</th><th>inf2</th><th>inf3</th><th colspan="2">Description</th></tr>
<tr><td>E_PAR</td><td rowspan="3">15</td><td>Indeterminate</td><td rowspan="3">Error regarding to registration of memory object (memory_object[])</td><td>1. Start address is not 16-bytes boundary.
2. bit15 of either acptn1, acptn2 and acptn3 is 1.
3. acptn1 == acptn2 == acptn3 == 0
4. Bits corresponded to the domain ID that is larger than the maximum domain ID of either acptn1, acptn2 and acptn3 is 1.
5. Start address > termination address</td></tr>
<tr><td>E_OBJ</td><td>Indeterminate</td><td>Multiple memory object with the same address are registered.</td></tr>
<tr><td>E_OACV</td><td>Indeterminate</td><td>The number of memory objects from which the access is permitted to same domain exceeds 7</td></tr>
<tr><td>E_PAR</td><td>16</td><td>Indeterminate</td><td>Error regarding to task creation (task[])</td><td>The "termination address of user stack + 1" is not 16-bytes boundary.</td></tr>
</table>

(7) type == -16 (Error by undefined variable vector interrupt)

Table 6.15 Arguments Passed to System-Down Routine (type == -16)

inf1	inf2	inf3
(a) "-U" option is not specified for cfg600px Indeterminate (b) "-U" option is specified for cfg600px Vector number	Value of PC which is pushed to stack by CPU at interruption	Value of PSW which is pushed to stack by CPU at interruption

(8) type == -17 (Error by undefined fixed vector interrupt)

Table 6.16 Arguments Passed to System-Down Routine (type == -17)

inf1	inf2	inf3
(a) "-U" option is not specified for cfg600px Indeterminate (b) "-U" option is specified for cfg600px Vector number	Value of PC which is pushed to stack by CPU at interruption	Value of PSW which is pushed to stack by CPU at interruption

(9) type > 0(Error by vsys_dwn and ivsys_dwn)

0 or negative value for type has been reserved by the kernel Please use positive value for type when vsys_dwn and ivsys_dwn is used.

Table 6.17 Arguments Passed to System-Down Routine (type > 0)

inf1	inf2	inf3
Arguments specified for vsys_dwn or ivsys_dwn		

6.8.3 CPU State at Start

The system-down routine is executed in supervisor mode.

Table 6.18 PSW at Start of Access Exception Handler

No.	Bit	Initial value	Description
1	IPL	Same before system dwon	
2	I	0	All interrupts are masked.
3	PM	0	Supervisor mode
4	U	0	ISP (interrupt stack pointer)
5	C,Z,S,O	Indeterminate	
6	Other bits	0	

6.9 Precautions to Take when Using Floating-Point Arithmetic Instructions

It is only when the -fpu option is specified that the compiler outputs floating-point arithmetic instructions.

If the -chkfpu option is specified in the assembler, the floating-point arithmetic instructions written in a program are detected as warning.

(1) When using floating-point arithmetic instructions in tasks

Make settings that include the FPSW in system.context of the cfg file.

The initial value of the FPSW is 0x00000100. Please initialize FPSW if necessary.

(2) When using floating-point arithmetic instructions in handlers

It is necessary that the handler explicitly guarantee the FPSW register. And the initial value of the FPSW is undefined. To guarantee and initialize the FPSW register, write a program as shown in Figure 6.8.

```
#include <machine.h>          // To use the compiler-supplied intrinsic function
                              // get_fpsw(), set_fpsw(), includes machine.h
#include "kernel.h"
#include "kernel_id.h"
void handler(void)
{
    unsigned long old_fpsw;    // Declares variable for saving the FPSW register
    old_fpsw = get_fpsw();      // Saves the FPSW register
    set_fpsw(0x00000100);       // Initialize FPSW if necessary
    /* Floating-point arithmetic operation */
    set_fpsw(old_fpsw);         // Restores the FPSW register
}
```

Figure 6.8 Program Example for a Handler that Uses Floating-Point Arithmetic Instructions

6.10 Precautions to Take when Using a Microcomputer that Supports the DSP Function

When a microcomputer which support the DSP function is used, it is necessary to note the treatment of the ACC register (accumulator)

Concretely, please note it as follows when you use the following instructions which update ACC register.

MACHI, MACLO, MULHI, MULLO, RACW, MVTACHI, MVTACLO

In no case does the compiler generate these instructions.

Note also that if the -chkdsp option is specified in the assembler, the DSP function instructions written in a program are detected as warning.

(1) When using the above-mentioned instructions in tasks

Make settings that include ACC in system.context of the cfg file.

(2) Guarantee of ACC register by Interrupt Handlers

If the application contains any tasks or interrupt handlers that use the above-mentioned instructions, it is necessary that all of the interrupt handlers guarantee the ACC register. There are the following three method.

1. Use "save_acc" compiler option
2. Specify "ACC" for "pragma_switch" at definition of all interrupt handlers in the cfg file.
3. All interrupt handlers explicitly guarantee the ACC register
   Figure 6.9 shows an example of how to write a handler that guarantees the ACC register.

```
#include "kernel.h"
#include "kernel_id.h"

typedef struct {                                // Structure to save the ACC register
    unsigned long upp;   // bit63-32
    unsigned long mid;   // bit47-16
} ST_ACC;

#pragma inline_asm(get_acc)                     // Macro to save the ACC register
void get_acc(ST_ACC *pk_acc)
{
    mvfachi r5
    mov.l   r5,[r1]
    mvfacmi r5
    mov.l   r5,4[r1]
}

#pragma inline_asm(set_acc)                    // Macro to restore the ACC register
void set_acc(ST_ACC *pk_acc)
{
    mov.l   [r1],r5
    mvtachi r5
    mov.l   4[r1],r5
    shll    #16,r5
    mvtaclo r5
}
void handler(void)                 // Handler function
{
    ST_ACC st_acc;                 // Declares variable for saving the ACC register
    get_acc(&st_acc);              // Saves the ACC register
     /* processing */
    set_acc(&st_acc);              // Restores the ACC register
}
```

Figure 6.9 Example of a Handler that Guarantees the ACC Register

7. Procedure for Generating the Load Module

7.1 Summary

The application programs for the RI600/PX are generally developed following the procedure described below.

(1) Generating a project

To use High-performance Embedded Workshop, create a new project that uses the RI600/PX in High-performance Embedded Workshop.

(2) Coding the application program

Code the application program. Correct the sample startup program as necessary.

(3) Creating a configuration file (cfg file)

Using a text editor, etc., create a cfg file that defines task entry addresses and stack sizes. The GUI configurator may also be used to create a cfg file.

(4) Executing the command line configurator (cfg600px)

The cfg600px loads the cfg file, from which it generates system data definition files (e.g., kernel_rom.h, kernel_ram.h) and include files for the application (e.g., kernel_id.h).

(5) Generating the load module

Execute the make command or execute a build in High-performance Embedded Workshop to generate the load module.

Figure 7.1 shows a flowchart, following which a load module is generated.

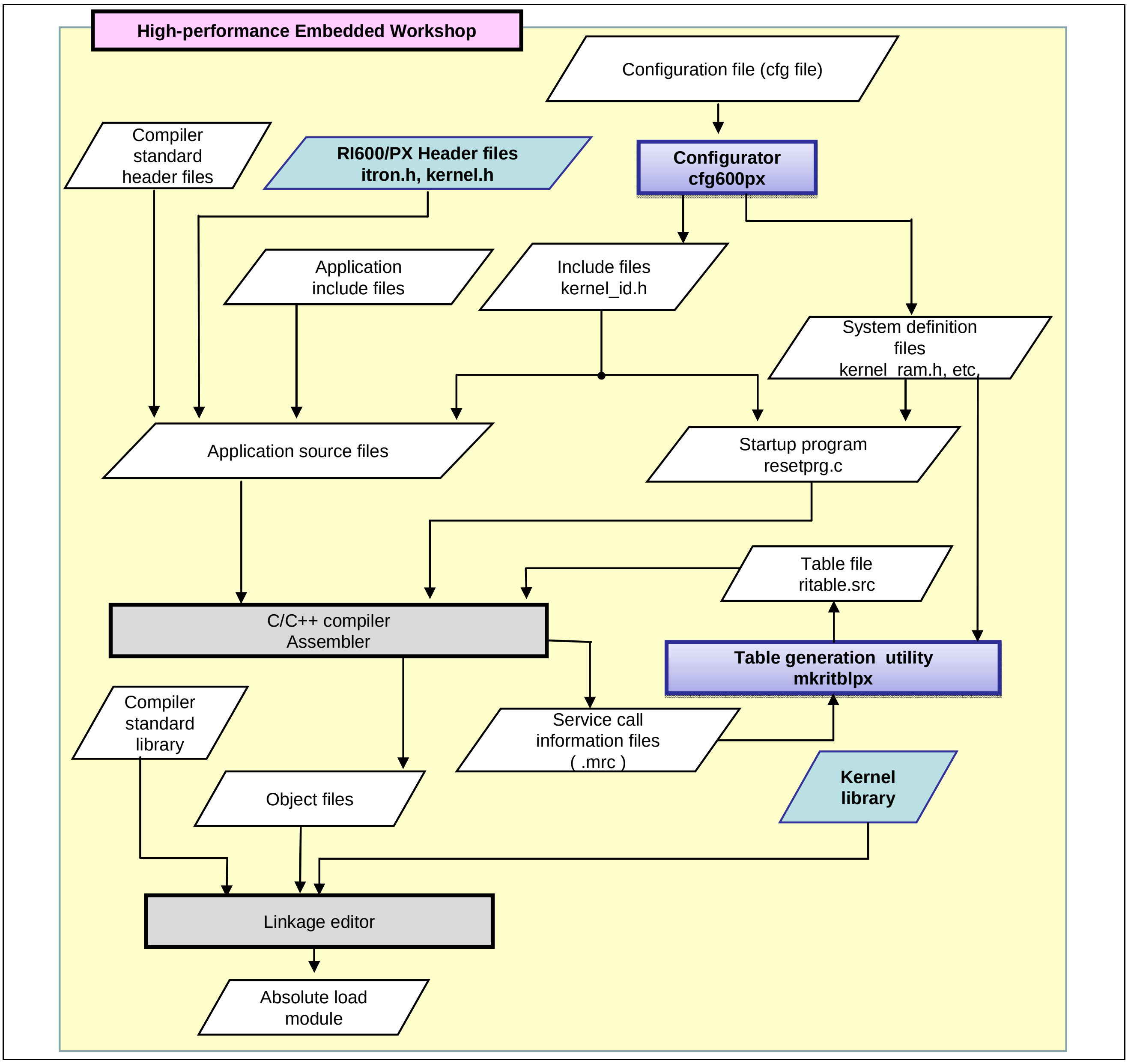

Figure 7.1 Flowchart of Load Module Generation

7.2 Creating Startup File (resetprg.c)

The startup file includes following statements. Note, it usually makes it to resetprg.c though the file name is arbitrary.

(1) Startup program (PowerON_Reset_PC())
(2) Timer initialization call-back function (_RI_init_cmt_knl())
(3) Access exception handler (_RI_sys_acccess_exception())
(4) System-down routine (_RI_sys_dwn__(())
(5) Getting kernel_rom.h and kernel_ram.h

(1) Startup Program (PowerON_Reset_PC())

The startup program is the program which is register in the reset vector. The startup program executes in the supervisor mode.

The startup program usually does the following processing.

- Initializes the processor and hardware
  If the high-speed interrupt function is used, it is necessary to initialize FINTV register.
- Initializes the execution environment for C/C++ language software (for example, by initializing sections)
- Start kernel (call vsta_knl or ivsta_knl())

Please maintain the state that any interrupt is not accepted until calling vsta_knl. Note, PSW.I at reset is 0 (all interrupts are masked).

The function name of the startup program is usually made "void PowerON_Reset_PC(void)". It is necessary to define the function name in interrupt_fvector[31] when assuming the function name excluding this.

It is necessary to declare "#pragma entry" directive for startup program. The compiler generates the object code to initialize stack pointer (ISP) to the system stack (SI section) by this declaration and "#pragma stacksize" directive in the kernel_ram.h which is generated by cfg600px.

(2) Timer Initialization Call-back Function (_RI_init_cmt_knl())

This function is called from vsta_knl and ivsta_knl.

When CMT0, CMT1, CMT2 or CMT3 is specified for clock.timer, includes this function should call in-line function "void _RI_init_cmt(void)" which is defined in the ri_cmt.h file which is generated by cfg600px.

When NOTIMER is specified for clock.timer, this function should return immediately.

This function is called in the state that all interrupts are masked. Please maintain this state.

(3) Access Exception Handler (_RI_sys_acccess_exception())

Refer to 6.7 "Access Exception Handler."

(4) System-Down Routine (_RI_sys_dwn__())

Refer to 6.8, "System-Down Routine."

(5) Getting kernel_rom.h and kernel_ram.h Included

The kernel_rom.h and kernel_ram.h are the system definition files generated by cfg600px. These files contain definitions of various data areas, etc.

The startup file (resetprg.c) must include these files in the following order.

```
#include "kernel.h"          /* provided by RI600/PX */
#include "kernel_id.h"      /* generated by cfg600px */
#include "kernel_ram.h"     /* generated by cfg600px */
#include "kernel_rom.h"     /* generated by cfg600px */
```

(6) Compiler Option

It is necessary to specify "-nostuff" option for the startup file.

(7) Example of the Startup File

```
#include    <machine.h>
#include    <_h_c_lib.h>
//#include  <stddef.h>                  // Remove the comment when you use errno
//#include  <stdlib.h>                  // Remove the comment when you use rand()
#include    "typedefine.h"              // Define Types
#include    "kernel.h"                  // Provided by RI600/PX
#include    "kernel_id.h"               // Generated by cfg600px

#if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))
#include    "ri_cmt.h"                  // Generated by cfg600px
                                        // Do comment-out when clock.timer is either NOTIMER or OTHER.
#endif

#ifdef __cplusplus
extern "C" {
#endif
void PowerON_Reset_PC(void);
void _RI_init_cmt_knl(void);
void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr);
void _RI_sys_dwn__( W type, VW inf1, VW inf2, VW inf3 );
#ifdef __cplusplus
}
#endif

// #ifdef __cplusplus                   // Use SIM I/O
// extern "C" {
// #endif
// extern void _INIT_IOLIB(void);
// extern void _CLOSEALL(void);
// #ifdef __cplusplus
// }
// #endif

#define FPSW_init 0x00000000  // FPSW bit base pattern

// extern void srand(_UINT);      // Remove the comment when you use rand()
// extern _SBYTE *_slptr;         // Remove the comment when you use strtok()

// #ifdef __cplusplus                   // Use Hardware Setup
// extern "C" {
// #endif
// extern void HardwareSetup(void);
// #ifdef __cplusplus
// }
// #endif

// #ifdef __cplusplus             // Remove the comment when you use global class object
// extern "C" {                   // Sections C$INIT and C$END will be generated
// #endif
// extern void _CALL_INIT(void);
// extern void _CALL_END(void);
// #ifdef __cplusplus
// }
// #endif

////////////////////////////////////////////////////////////
// Section definition
////////////////////////////////////////////////////////////
#pragma section P PS
#pragma section B BS
#pragma section C CS
```

```
#pragma section D DS

#pragma entry PowerON_Reset_PC

////////////////////////////////////////////////////////////
// Power-on Reset Program
////////////////////////////////////////////////////////////
void PowerON_Reset_PC(void)
{
#ifdef __ROZ                                // Initialize FPSW
#define _ROUND 0x00000001                   // Let FPSW RMbits=01 (round to zero)
#else
#define _ROUND 0x00000000                   // Let FPSW RMbits=00 (round to nearest)
#endif
#ifdef __DOFF
#define _DENOM 0x00000100                   // Let FPSW DNbit=1 (denormal as zero)
#else
#define _DENOM 0x00000000                   // Let FPSW DNbit=0 (denormal as is)
#endif
    set_fpsw(FPSW_init | _ROUND | _DENOM);
#endif

    _INITSCT();                     // Initialize Sections

//  _INIT_IOLIB();                  // Use SIM I/O

//  errno=0;                        // Remove the comment when you use errno
//  srand((_UINT)1);                // Remove the comment when you use rand()
//  _s1ptr=NULL;                    // Remove the comment when you use strtok()

//  HardwareSetup();                // Use Hardware Setup
    nop();

//  set_fintv(<handler address>);   // Initialize FINTV register

//  _CALL_INIT();                   // Remove the comment when you use global class object

    vsta_knl();                     // Start RI600/PX
                                    // Never return from vsta_knl

//  _CLOSEALL();                    // Use SIM I/O

//  _CALL_END();                    // Remove the comment when you use global class object

    brk();

}

////////////////////////////////////////////////////////////
// Timer initialize call-back
////////////////////////////////////////////////////////////
void _RI_init_cmt_knl(void)
{
#if (((_RI_CLOCK_TIMER) >=0) && ((_RI_CLOCK_TIMER) <= 3))  // Do comment-out when clock.timer is either NOTIMER or OTHER.
    _RI_init_cmt();
#endif
}

////////////////////////////////////////////////////////////
// Access exception handler
////////////////////////////////////////////////////////////
void _RI_sys_access_exception(UW pc ,UW psw, UW sts, UW addr)
{
```

```
    // Now PSW.I=0 (all interrupts are masked.)

    ID  hTaskID;

    iget_tid(&hTaskID);
    iras_tex(hTaskID, 1);
}

////////////////////////////////////////////////////////////
// System-down routine for RI600/PX
////////////////////////////////////////////////////////////
struct SYSDWN_INF{
    W   type;
    VW  inf1;
    VW  inf2;
    VW  inf3;
};

volatile struct SYSDWN_INF _RI_sysdwn_inf;

void _RI_sys_dwn__( W type, VW inf1, VW inf2, VW inf3 )
{
    // Now PSW.I=0 (all interrupts are masked.)

    _RI_sysdwn_inf.type = type;
    _RI_sysdwn_inf.inf1 = inf1;
    _RI_sysdwn_inf.inf2 = inf2;
    _RI_sysdwn_inf.inf3 = inf3;

    while(1)
    ;
}

/////////////////////////////////////////////////////////////////////////////
// RI600/PX system data
/////////////////////////////////////////////////////////////////////////////
#include "kernel_ram.h"     // generated by cfg600px
#include "kernel_rom.h"     // generated by cfg600px
```

7.3 Kernel Libraries

The kernel library consists of ri600lit.lib for use in little-endian form and ri600big.lib for use in big-endian form. These libraries are stored in the "<installation directory>\lib600."

Please use either of libraries according to endian type of the MCU.

7.4 Section List

7.4.1 Naming Convention of Section

Usually, a name unique in each section is given. The following naming convention is recommended to be provided to facilitate this.

(1) 1st Character : Section Type

- P : Program area
- C : Constant area
- B : Uninitialized data area
- D : Initialized data area (ROM)
- R : Variables area for initialized data (RAM) (generated by linker)
- W : Switch statement branch table area (generated by compiler)
- L : Literal area (generated by compiler)

(2) Since the 2nd Character

- RI* : Reserved by RI600/PX
  This area is not accessed from user mode (= task comtext).
- U* : Memory object or user stack
  This area is accessed from user mode (= task comtext).
- S* : Excluding the above-mentioned
  This area is not accessed from user mode (= task comtext).

7.4.2 List of RI600/PX Sections

Table 7.1 List of RI600/PX Sections

Section	Description	Attribute	Boundary alignment
PRI_KERNEL	Kernel program	CODE	1
CRI_ROM	Kernel constant	ROMDATA	4
DRI_ROM	Kernel initialized data	ROMDATA	4
RRI_RAM	Variables area for kernel initialized data	DATA	4
FIX_INTERRUPT_VECTOR	Fixed interrupt vector table This section must be mapped to the address 0xFFFFFF80.	ROMDATA	4
INTERRUPT_VECTOR	Variable interrupt vector table	ROMDATA	4
SI	System stack	DATA	4
SURI_STACK	The section name assigned to user stack for tasks can be specified in the cfg file. When this is omitted, SURI_STACK is applied as the section name.	DATA	4
BRI_RAM	Kernel uninitialized data The section name assigned to message buffer area can be specified in the cfg file. When this is omitted, BRI_RAM is applied as the section name.	DATA	4
BURI_HEAP	The section name assigned to fixed-sized memory pool and variable-sized memory pool area can be specified in the cfg file. When this is omitted, BURI_HEAP is applied as the section name. This may be included in memory object if necessary.	DATA	4

7.4.3 "aligned_section" Option for Linker

It is necessary to specify "aligned_section" option for the following section at linking.

1. Section specified in the "memory_object[].start_address"
2. Section specified in the "task[].stack_section"
3. SURI_STACK

7.4.4 Attention Concerning L and W section

The L section is literal area, and the W section is switch statement branch table area. These section are generated by compiler.

The name of these section cannot be changed by using "#pragma section" directive.

Please note it as follows when a function executes as a task and the L and W section may be generated by compiling the source file.

(1) All function in the source file executes only as tasks which belong to same domain

Especially, there are no points of concern. The L and W section should be a memory object to be able to read from the domain.

(2) All function in the source file executes as tasks which belong to separate domains

Because a different name cannot be given to literal area and switch table area of each function, it is impossible to divide these area to separate memory object. Therefore, separate functions in order to the domain at running to individual source file and apply (1), or the L and W section should be a memory object to be able to read from all domain.

7.5 Service Call Information File (mrc file) and Essential Compiler Option

Service call information files (mrc files) are generated by compiling application source files including "kenrel.h".

The name of service calls that the source file uses is output to mrc file. The mrc file should be inputted to mkritblpx.

The "-ri600_preinit_mrc" compiler option should be specified for application files which includes "kernel.h". Service call modules that application does not use might be linked though the problem is not caused in run-time even if this option is not specified.

Please input mrc files generated by compiler to mkritblpx when you make library. Make a mrc file that describes service call name and input the mrc file to mkritblpx when it is difficult to input mrc files generated by compiler or you want to link service calls that are used by application program and the application program is not linked with the kernel (refer to the following).

Note, when unlinked service call is issued, the system will go down.

```
sta_tsk
snd_mbx
rcv_mbx
prcv_mbx
```

7.6 Notes

7.6.1 Processor mode

Tasks and task exception handling routines are executed in user mode (PSW.PM=1). Handlers and the kernel are executed in supervisor mode (PSW.PM=0).

The CPU detects privileged instruction exception when a privileged instruction is executed in user mode. In the program executed in user mode, please do not use a privileged instruction.

The assembler supports "-chkpm" option which detects privileged instruction as warning, and uses this option if necessary.

7.6.2 Address 0

An error will occur when 0 is specified for a service call parameter except start address of a memory object. Therefore, do not allocate following sections to address 0.

1. BURI_STACK
2. BURI_HEAP
3. task[].stack_section
4. message_buffer[].mbf_section
5. memorypool[].section
6. variable_memorypool[].mpl_section

8. Configurator (cfg600px)

8.1 Creating a Configuration File (cfg File)

The OS resources used in the application must be registered with the RI600/PX system. A configuration file (cfg file) is where these settings are made, and the tool used for registration with the system is the configurator (cfg600px).

Based on the content defined in a configuration file (cfg file), cfg600px generates the files needed to build the kernel.

8.2 Representation Format in cfg File

This section describes the representation format of the definition data in the cfg file.

(1) Comment Statement

A statement from a double slash (//) to the end of a line is handled as a comment and no processing is applied.

(2) End of Statement

A statement must end with a semicolon (;).

(3) Numeric Value

A numeric value must be written in one of the following formats.

- Hexadecimal
  Add "0x" or "0X" at the beginning of a numeric value or add "h" or "H" at the end. In the latter format, be sure to add "0" at the beginning when the value begins with an alphabetic letter from A to F or a to f. Note that the configurator does not distinguish between uppercase and lowercase letters for alphabetic letters (A to F or a to f) used in numeric value representation.[5]
- Decimal
  Simply write an integer value as is usually done (23, for example). Note that a decimal value must not begin with "0".
- Octal
  Add "0" at the beginning of a numeric value or add "O" or "o" at the end.
- Binary
  Add "B" or "b" at the end of a numeric value. Note that a binary value must not begin with "0".

Table 8.1 Examples of Numeric Value Representation

Hexadecimal	0xf12 0Xf12 0a12h 0a12H 12h 12H
Decimal	32
Octal	017 17o 17O
Binary	101110b 101010B

[5] The configurator distinguishes uppercase and lowercase letters except for ‘A’ to ‘F’ and ‘a’ to ‘f’ in numeric value representation.

A numeric value can include operators. Table 8.2 shows the available operators.

Table 8.2 Operators

Operator	Precedence	Direction of Computation
()	High	Left to right
(unary minus)		Right to left
* / %		Left to right
+ - (binary minus)	Low	Left to right

The following are examples of numeric values.

- 123
- 123 + 0x23
- (23/4 + 3) * 2
- 100B + 0aH

A numeric value greater than 0xFFFFFFFF must not be specified.

(4) Symbol

A symbol is a string of numeric characters, uppercase alphabetic letters, lowercase alphabetic letters, and underscores (_). It must not begin with a numeric character.

The following are examples of symbols.

- _TASK1
- IDLE3

(5) Function Name

A function name consists of numeric characters, uppercase alphabetic letters, lowercase alphabetic letters, underscores (_), and dollar signs ($). It must not begin with a numeric character and must end with "()".

The following are examples of function names.

- main()
- func()

To specify module name written by assembly language, name to start the start label with '_', and specify the name that excludes '_' for function name.

(6) Frequency

The frequency is indicated by a character string that consist of numerals and . (period), and ends with MHz. The numerical values are significant up to six decimal places. Also note that the frequency can be entered using decimal numbers only.

Frequency entry examples are presented below.

- 16MHz
- 8.1234MHz

It is also well to remember that the frequency must not begin with . (period).

8.3 Default cfg File

For most definition items, if the user omits settings, the settings in the default cfg file are used.

The default cfg file is stored in the directory indicated by environment variable "LIB600". Be sure not to edit this file.

8.4 Definition Items in cfg File

The following items should be defined in the cfg file.

- System definition (system)
- System clock definition (clock)
- Maximum ID definition (maxdefine)
- Domain definition (domain[])
- Memory object definition (memory_object[])
- Task definition (task[])
- Semaphore definition (semaphore[])
- Eventflag definition (flag[])
- Data queue definition (dataqueue[])
- Mailbox definition (mailbox[])
- Mutex definition (mutex[])
- Message buffer definition (message_buffer[])
- Fixed-sized memory pool definition (memorypool[])
- Variable-sized memory pool definition (variable_memorypool[])
- Cyclic handler definition (cyclic_hand[])
- Alarm handler definition (alarm_hand[])
- Relocatable interrupt vector definition (interrupt_vector[])
- Fixed interrupt vector definition (interrupt_fvector[])

8.4.1 System Definition (system)

Here, define the general information relating to the kernel system. In this definition, system cannot be omitted.

❑Format

```
system {
        stack_size          = (1) System stack size;
        priority            = (2) Maximum value of task priority;
        system_IPL          = (3) Kernel interrupt mask level;
        message_pri         = (4) Maximum value of message priority;
        tic_deno            = (5) Time tick denominator;
        tic_nume            = (6) Time tick numerator;
        context             = (7) Task context register;
};
```

❑Contents

(1) System stack size (stack_size)

Description: Define the total stack size used in service call processing and interrupt processing.
Definition format: Numeric value
Definition range: Multiple of 4 of 8 or more
When omitting: The set value in default cfg file (factory setting: 0x800) applied

(2) Maximum value of task priority (priority)

Description: Define the maximum value of task priority used in the application.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The set value in default cfg file (factory setting: 32) applied

(3) Kernel interrupt mask level (system_IPL)

Description: Define the interrupt mask level when the kernel's critical section is executed (PSW register's IPL value). Interrupts with higher priority levels than that are handled as "non-kernel" interrupts.
Definition format: Numeric value
Definition range: 1 - 15
When omitting: The set value in default cfg file (factory setting: 7) applied

(4) Maximum value of message priority (message_pri)

Description: Define the maximum value of message priority used in the mailbox function. Note that if the mailbox function is unused, this definition item has no effect.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The set value in default cfg file (factory setting: 255) applied

(5) Time tick denominator (tic_deno)

Description: Define the denominator of the time tick. At least one of the time tick numerator or denominator must be 1.
The time tick time (kernel timer interrupt cycle) is calculated by the equation below.

Time tick time (in millisecond) = tic_nume / tic_deno

No matter how tic_nume and tic_deno are set, the units of time in which units service calls are handled is always milliseconds (millisecond). It is by tic_nume and tic_deno that the accuracy of the time that the kernel manages is determined.
Definition format: Numeric value
Definition range: 1 - 100
When omitting: The set value in default cfg file (factory setting: 1) applied

(6) Time tick numerator (tic_nume)

Description: Define the numerator of the time tick. For details, see the preceding item.
Definition format: Numeric value
Definition range: 1 - 65535
When omitting: The set value in default cfg file (factory setting: 1) applied

(7) Task context register (context)

Description: Define the register set used by tasks and task exception handling routines. The settings made here apply to all tasks and task exception handling routines.
Definition format: Symbol
Definition range: Select one from Table 8.3.

Table 8.3 system.context

Set value	Registers guaranteed as task context			
	CPU		FPU	DSP
	PSW, PC, R0 - R7, R14, R15	R8 - R13	FPSW	ACC
NO	Guaranteed	Guaranteed	Not guaranteed	Not guaranteed
FPSW	Guaranteed	Guaranteed	Guaranteed	Not guaranteed
ACC	Guaranteed	Guaranteed	Not guaranteed	Guaranteed
FPSW,ACC	Guaranteed	Guaranteed	Guaranteed	Guaranteed
MIN	Guaranteed	Not guaranteed	Not guaranteed	Not guaranteed
MIN,FPSW	Guaranteed	Not guaranteed	Guaranteed	Not guaranteed
MIN,ACC	Guaranteed	Not guaranteed	Not guaranteed	Guaranteed
MIN,FPSW,ACC	Guaranteed	Not guaranteed	Guaranteed	Guaranteed

When omitting: The set value in default cfg file (factory setting: NO) applied
Remark: For system.context, please be sure to see Section 8.4.2, "Precautions to Take when Defining system.context."

8.4.2 Precautions to Take when Defining system.context

(1) Precautions regarding the FPU and ACC (accumulator)

The value to be set for system.context differs depending on how the FPU and DSP are handled.

Please set system.context appropriately according to the following explanations.

Note, if system.context is set to other than recommended value, the kernel performance may be slightly deteriorated, compared to the recommended settings case.

Remark: The compiler outputs floating-point arithmetic instructions only when the "-cpu=rx600" and "-fpu" options are specified. If the -chkfpu option is specified in the assembler, the floating-point arithmetic instructions written in a program are detected as warning.
In no case does the compiler output the DSP function instructions. If the -chkdsp option is specified in the assembler, the DSP function instructions written in a program are detected as warning.

(a) When using a microcomputer that incorporates the FPU and the DSP (accumulator)

Table 8.4 When using a microcomputer that incorporates the FPU and the DSP (accumulator)

Usage condition of instruction in tasks and task exception handling routines		system.context
Floating-point arithmetic instructions	DSP function instructions	
Yes	Yes	FPSW- and ACC- included settings essential
Yes	No	FPSW-included setting essential and ACC-excluded setting recommended
No	Yes	ACC-included setting essential and FPSW-excluded setting recommended
No	No	FPSW- and ACC- excluded settings recommended

(b) When using a microcomputer that incorporates the FPU, but does not contain the DSP (accumulator)

Table 8.5 When using a microcomputer that incorporates the FPU, but does not contain the DSP (accumulator)

Usage condition of instruction in tasks and task exception handling routines		system.context
Floating-point arithmetic instructions	DSP function instructions	
Yes	Yes	(DSP function instructions cannot be used since the MCU does not have DSP.)
Yes	No	FPSW-included and ACC-excluded settings essential
No	Yes	(DSP function instructions cannot be used since the MCU does not have DSP.)
No	No	ACC-excluded setting essential and FPSW-excluded setting recommended

(c) When using a microcomputer that does not incorporate the FPU, but contains the DSP (accumulator)

Table 8.6 When using a microcomputer that does not incorporate the FPU, but contains the DSP (accumulator)

Usage condition of instruction in tasks and task exception handling routines		system.context
Floating-point arithmetic instructions	**DSP function instructions**	
Yes	Yes	(Floating-point arithmetic instructions cannot be used since the MCU does not have FPU.)
Yes	No	
No	Yes	FPSW-excluded and ACC-included settings essential
No	No	ACC-excluded setting essential and FPSW-excluded setting recommended

(d) When using a microcomputer that incorporates neither the FPU nor the DSP (accumulator)

Table 8.7 When using a microcomputer that incorporates neither the FPU nor the DSP (accumulator)

Usage condition of instruction in tasks and task exception handling routines		system.context
Floating-point arithmetic instructions	**DSP function instructions**	
Yes	Yes	(Floating-point arithmetic instructions and DSP function instructions cannot be used since the MCU does not have FPU and DSP.)
Yes	No	
No	Yes	
No	No	FPSW- and ACC-excluded settings essential

(2) Relationship with the compiler options "fint_register", "base" and "pid"

In system.context, by selecting one of choices "MIN," "MIN, ACC," "MIN, FPSW," or "MIN, ACC, FPSW," it is possible to configure the registers so that R8–R13 registers will not be saved as task contexts. This results in an increased processing speed.

Note, however, that such a setting of system.context is permitted in only the case where all of R8–R13 registers are specified to be used by the compiler options "fint_register", "base" and "pid". For example, the following cases of option specification apply to this:

Good example:

Example 1: -fint_register=4 -base=rom=R8 -base=ram=R9
Example 2:-fint_register=3 -base=rom=R8 -base=ram=R9 -base=0x80000=R10

If, in any other case, the above setting is made for system.context, the kernel will not operate normally. For example, the following cases of option specification apply to this:

Bad example:

Example 3: No fint_register and base options
Example 4: -fint_register=4
Example 5: -base=rom=R8 -base=ram=R9
Example 6: -fint_register=3 -base=rom=R8 -base=ram=R9

8.4.3 System Clock Definition (clock)

Define the clock frequency and other information relating to the system clock.

❑Format

```
clock {
        timer           = (1) Selection of the system timer;
        template        = (2) Template file;
        timer_clock     = (3) System timer clock frequency;
        IPL             = (4) Timer interrupt priority level;
};
```

❑Contents

(1) Selection of the system timer (timer)

Description: Define the hardware timer used for the system clock.
Definition format: Symbol
Definition range: Select one out of the following. If one of CMT0, CMT1, CMT2, or CMT3 is specified, the cfg600px generates the timer driver source code (ri_cmt.h).

- CMT0: Uses the microcomputer's internal CMT channel 0.
- CMT1: Uses the microcomputer's internal CMT channel 1.
- CMT2: Uses the microcomputer's internal CMT channel 2.
- CMT3: Uses the microcomputer's internal CMT channel 3.
- OTHER: Uses a timer other than the above. In this case, the user needs to create a timer initialize routine.
- NOTIMER: Does not use the system timer function.

When omitting: The set value in default cfg file (factory setting: CMT0) applied

(2) Template file (template)

Description: Specify template file where hardware information and initialization function of CMT is described.
This definition is ignored when one of NOTIMER or OTHER is specified for clock.timer.
The template files are provided by RI600/PX. Please refer to the release note of the product attachment for the delivered template files and relationship between template files and MCU type No.
Either CMT0, CMT1, CMT2 or CMT3 might be unsupported according to template file. When unsupported CMT channel is specified for clock.timer, the cfg600px does not detect error but ri_cmt.h detect error at compilation.
Definition format: Symbol
Definition range: -
When omitting: The set value in default cfg file (factory setting: (rx630.tpl) applied

(3) System timer clock frequency (timer_clock)

Description: Define the frequency of the clock supplied to the system timer.
If one of CMT0, CMT1, CMT2, or CMT3 is selected for timer, specify the frequency of the PCLK (peripheral module clock).
Definition format: Frequency
Definition range: -
When omitting: The set value in default cfg file (factory setting: 25MHz) applied

(4) Timer interrupt priority level (IPL)

Description: Define the interrupt priority level of the system timer. Interrupts with lower priority levels than the one defined here are not accepted while the system timer interrupt handler is being executed.
Definition format: Numeric value
Definition range: 1 - system.system_IPL
When omitting: The set value in default cfg file (factory setting: 4) applied

(5) Points of concern for timer=OTHER

The following actions are required.

1. Initialize the timer before starting the kernel (vsta_knl).
   The cfg600px outputs following macros to the "kernel_id.h". Please initialize the timer based on this information.
   - _RI_CLOCK_IPL : Timer interrupt priority level (clock.IPL)
   - TIC_DENO : Time tick denominator (system.tic_deno)
   - TIC_NUME : Time tick numerator (system.tic_nume)
2. Define the relocatable interrupt vector as follows.

```
interrupt_vector[<Vector number>] {
    entry_address = __RI_SYS_STMR_INH;
    os_int = YES;
};
```

8.4.4 Maximum ID Definition (maxdefine[])

The definition item maxdefine is provided for the definition of the maximum ID for each domain. And this definition is required to use service calls to create an object dynamically.

The macros in which the maximum ID of each object are defined is output to kernel_id.h. (Refer to "4.2.1 Constant Macro")

❑Format

```
maxdefine {
        max_task           = (1) Maximum task ID;
        max_sem            = (2) Maximum semaphore ID;
        max_flag           = (3) Maximum eventflag ID;
        max_dtq            = (4) Maximum data queue ID;
        max_mbx            = (5) Maximum mailbox ID;
        max_mtx            = (6) Maximum mutex ID;
        max_mbf            = (7) Maximum message buffer ID;
        max_mpf            = (8) Maximum fixed-sized memory pool ID;
        max_mpl            = (9) Maximum variable-sized memory pool ID;
        max_cyh            = (10) Maximum cyclic handler ID;
        max_alh            = (11) Maximum alarm handler ID;
        max_domain         = (12) Maximum domainID;
};
```

❑Contents

(1) Maximum task ID (max_task)

Description : The cre_tsk, acre_tsk, del_tsk, exd_tsk and def_tex can be used by this definition. Ranges of task ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_task", the ID number defined in "task[]" and the number of "task[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_tsk, acre_tsk, del_tsk and def_tex returns E_NOSPT error. And exd_tsk causes system-down.
Ranges of task ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "task[]" and the number of "task[]" definitions

(2) Maximum semaphore ID (max_sem)

Description : The cre_sem, acre_sem and del_sem can be used by this definition. Ranges of semaphore ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_sem", the ID number defined in "semaphore[]" and the number of "semaphore[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_sem, acre_sem and del_sem returns E_NOSPT error.
Ranges of semaphore ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "semaphore[]" and the number of "semaphore[]" definitions

(3) Maximum eventflag ID (max_flg)

Description : The cre_flg, acre_flg and del_flg can be used by this definition. Ranges of eventflag ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_flag", the ID number defined in "flag[]" and the number of "flag[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_flg, acre_flg and del_flg returns E_NOSPT error.
Ranges of eventflag ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "flag[]" and the number of "flag[]" definitions

(4) Maximum data queue ID (max_dtq)

Description : The cre_dtq, acre_dtq and del_dtq can be used by this definition. Ranges of data queue ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_dtq", the ID number defined in "data_queue[]" and the number of "data_queue[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_dtq, acre_dtq and del_dtq returns E_NOSPT error.
Ranges of data queue ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "data_queue[]" and the number of "data_queue[]" definitions

(5) Maximum mailbox ID (max_mbx)

Description : The cre_mbx, acre_mbx and del_mbx can be used by this definition. Ranges of mailbox ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_mbx", the ID number defined in "mailbox[]" and the number of "mailbox[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_mbx, acre_mbx and del_mbx returns E_NOSPT error.
Ranges of mailbox ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "mailbox[]" and the number of "mailbox[]" definitions

(6) Maximum mutex ID (max_mtx)

Description : The cre_mtx, acre_mtx and del_mtx can be used by this definition. Ranges of mutex ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_mtx", the ID number defined in "mutex[]" and the number of "mutex[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_mtx, acre_mtx and del_mtx returns E_NOSPT error.
Ranges of mutex ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "mutex[]" and the number of "mutex[]" definitions

(7) Maximum message buffer ID (max_mbf)

Description : The cre_mbf, acre_mbf and del_mbf can be used by this definition. Ranges of message buffer ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_mbf", the ID number defined in "message_buffer[]" and the number of " message_buffer[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_mbf, acre_mbf and del_mbf returns E_NOSPT error.
Ranges of message buffer ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in " message_buffer[]" and the number of " message_buffer[]" definitions

(8) Maximum fixed-sized memory pool ID (max_mpf)

Description : The cre_mpf, acre_mpf and del_mpf can be used by this definition. Ranges of fixed-sized memory pool ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_mpf", the ID number defined in "memorypool[]" and the number of "memorypool[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_mpf, acre_mpf and del_mpf returns E_NOSPT error.
Ranges of fixed-sized memory pool ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "memorypool[]" and the number of "memorypool[]" definitions

(9) Maximum variable-sized memory pool ID (max_mpl)

Description : The cre_mpl, acre_mpl and del_mpl can be used by this definition. Ranges of variable-sized memory pool ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_mpl", the ID number defined in "variable_memorypool[]" and the number of "variable_memorypool[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_mpl, acre_mpl and del_mpl returns E_NOSPT error.
Ranges of variable-sized memory pool ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "variable_memorypool[]" and the number of "variable_memorypool[]" definitions

(10) Maximum cyclic handler ID (max_cyh)

Description : The cre_cyc, acre_cyc and del_cyc can be used by this definition. Ranges of cyclic handler ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_cyh", the ID number defined in "cyclic_hand[]" and the number of "cyclic_hand[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_cyc, acre_cyc and del_cyc returns E_NOSPT error.
Ranges of cyclic handler ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "cyclic_hand[]" and the number of "cyclic_hand[]" definitions

(11) Maximum alarm handler ID (max_alh)

Description : The cre_alm, acre_alm and del_alm can be used by this definition. Ranges of alarm handler ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among "max_alh", the ID number defined in "alarm_hand[]" and the number of "alarm_hand[]" definitions

Definition format : Numeric value
Definition range: 1 - 255
When omitting: The cre_alm, acre_alm and del_alm returns E_NOSPT error.
Ranges of alarm handler ID that can be used are is as follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "alarm_hand[]" and the number of "alarm_hand[]" definitions

(12) Maximum domain ID (max_domain)

Description : Ranges of domain ID that can be used areas follows.

- Minimum value : 1
- Maximum value : The largest one among max_domain and the ID number defined in "domain[]"

Definition format : Numeric value
Definition range: 1 - 15
When omitting: Ranges of domain ID that can be used areas follows.

- Minimum value : 1
- Maximum value : The largest one among the ID number defined in "domain[]"

8.4.5 Domain Definition (domain[])

The definition item domain[] is provided for the definition of domains. The domain that is not defined by this statement is handled as "trust = NO".

❑Format

```
domain[(1) ID number]{
        trust              = (2) Trusted domain;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 15
When omitting: Cannot be omitted (error assumed)

(2) Trusted domain (trust)

Description: Define whether the domain is trusted or not.
Definition format: Symbol
Definition range: Select either of the following:

- YES : The domain is trusted.
- NO : The domain is not trusted.

When omitting: The set value in default cfg file (factory setting: YES) applied

8.4.6 Memory Object Definition (memory_object[])

The definition item memory_object[] is provided for the definition (registratuion)of memory objects.

At least one memry_object[] definition is necessary in the cfg file.

❑Format

```
memory_object[]{
        start_address       = (1) Start address of memory object;
        end_address         = (2) Termination address of memory object;
        acptn1              = (3) Access permission pattern for operand read;
        acptn2              = (3) Access permission pattern for operand write;
        acptn3              = (3) Access permission pattern for execution;
};
```

❑Contents

(1) Start address of memory object (start_address)

Description: Define start address of the memory object by numeric value or section name.

When section name is specified, the section must be allocated to 16-bytes boundary address at linking. This restriction is filled by specifying "aligned_section" option for this section at linking.

When numeric value is specified, the value must be multiple of 16.

Definition format: Symbol or numeric value
Definition range: 0 - 0xFFFFFFF0 and multiple of 16, when numeric value is specified
When omitting: Cannot be omitted (error assumed)

(2) Termination address of memory object (start_address)

Description: Define termination address of the memory object by numeric value or section name.

When section name is specified, the address in which termination address of this section is rounded up to "multiple of 16 + 15" is treated with the termination address of the memory object. If the termination address of this section is not "multiple of 16 + 15", you must not allocate any section from "the termination address of this section + 1" to next " multiple of 16 + 15" address.

When numeric value is specified, the value must be multiple of 16 + 15.

Definition format: Symbol or numeric value
Definition range: 0x0000000F - 0xFFFFFFFF and multiple of 16 + 15, when numeric value is specified
When omitting: Cannot be omitted (error assumed)

(3) Access permission pattern (acptn1, acptn2, acptn3)

Description: Define access permission pattern for operand read access (acptn1), operand write access (acptn2) and execution access (acptn3) by symbol or numeric value.
Only the "TACP_SHARED" (all domain can access) can be specified as symbol.
Use numeric value to specify permission for each domain according toFigure 8.1.

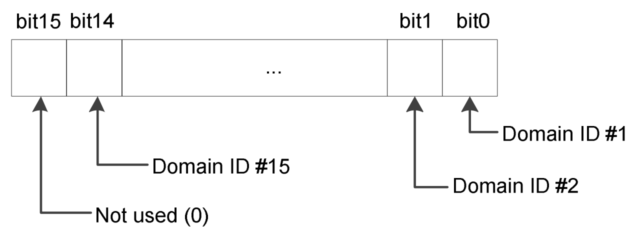

Figure 8.1 Access Permission Pattern

Definition format: Symbol or numeric value
Definition range: Symbol : TACP_SHARED (all domain can access)
Numeric value : 0 - 0x7FFF
When omitting: The set value in default cfg file (factory setting: TACP_SHARED) applied

8.4.7 Task Definition (task[])

The definition item task[] is provided for the definition (creation) of tasks.

Note that in specifications of the product described herein there isn't any particular task something like an initial startup task. When defining a task in the cfg file, specify ON for task[].inital_start, and the task will automatically goes to the READY state when the system starts. When multiple task[].initial_start are ON, they go to the READY state in small order of its ID number.

At least one task[] definition with initial_start=ON is necessary in the cfg file.

❑Format

```
task[(1) ID number]{
        name                 = (2) ID name;
        entry_address        = (3) Task entry address;
        stack_size           = (4) User stack size of task;
        stack_section        = (5) Section name assigned to the stack area;
        priority             = (6) Initial priority of task;
        initial_start        = (7) TA_ACT attribute (initial state after creation);
        exinf                = (8) Extended information;
        texptn               = (9) Task exception handling routine entry address;
        domain_num           = (10) Belonging domain ID;
};
```

❑Contents

(1) ID number

Description: Define the task ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Task entry address (entry_address)

Description: Define the entry address of the task starting from which it is executed.
Definition format: Function name
Definition range: -
When omitting: Cannot be omitted (error assumed)

(4) User stack size of task (stack_size)

Description: Define the user stack size of the task. The user stack refers to the stack area that each task uses. The RI600/PX requires that tasks be assigned a user stack individually.
The user stack area of a size specified here is generated by cfg600px.
Definition format: Numeric value
Definition range: More than value indicated in Table 8.8, and multiple of 16

Table 8.8 Lower bound of user stack size

system.context	Lower bound of user stack size
NO	68
FPSW	72
ACC	76
FPSW,ACC	80
MIN	44
MIN,FPSW	48
MIN,ACC	52
MIN,FPSW,ACC	56

When omitting: The set value in default cfg file (factory setting: 256) applied

(5) Section name assigned to the stack area (stack_section)

Description: Define the section name to be assigned to the user stack area. The section attribute is "DATA", and the alignment number is 4. When linking, be sure to allocate this section in the RAM area.
This section must not be in a memory object, and must not be allocated to address 0.
Definition format: Symbol
Definition range: -
When omitting: The set value in default cfg file (factory setting: SURI_STACK) applied

(6) Task initial priority (priority)

Description: Define the task initial priority.
Definition format: Numeric value
Definition range: 1 - system.priority
When omitting: The set value in default cfg file (factory setting: 1) applied

(7) TA_ACT attribute (initial state after creation) (initial_start)

Description: Define the initial state of the task that is either the READY state or a DORMANT state. This definition item corresponds to the task's TA_ACT attribute.
Definition format: Symbol
Definition range: Select either of the following:

- ON : Placed in the READY state at kernel startup
- OFF : Placed in DORMANT state at kernel startup

When omitting: The set value in default cfg file (factory setting: OFF) applied

(8) Extended information (exinf)

Description: Define the extended information of the task.
Definition format: Numeric value
Definition range: 0 - 0xFFFFFFFF
When omitting: The set value in default cfg file (factory setting: 0) applied

(9) Task exception handling routine entry address (texrtn)

Description: Define the entry address of the task exception handling routine starting from which it is executed.
Definition format: Function name
Definition range: -
When omitting: The task exception handling routine is not defined.

(10) Domain ID number

Description: Define the ID number of the domain by which the task belongs.
Definition format: Numeric value
Definition range: 1 - 15
When omitting: The set value in default cfg file (factory setting: 1) applied

8.4.8 Semaphore Definition (semaphore[])

The definition item semaphore[] is provided for the definition (creation) of semaphores.

❑Format

```
semaphore[(1) ID number]{
        name                = (2) ID name;
        max_count           = (3) Maximum value of semaphore counter;
        initial_count       = (4) initial value of semaphore counter;
        wait_queue          = (5) Queue attribute;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Maximum value of semaphore counter (max_count)

Description: Define the maximum value of the semaphore counter.
Definition format: Numeric value
Definition range: 1 - 65535
When omitting: The set value in default cfg file (factory setting: 1) applied

(4) Initial value of semaphore counter (initial_count)

Description: Define the initial value of the semaphore counter.
Definition format: Numeric value
Definition range: 0 to max_count
When omitting: The set value in default cfg file (factory setting: 1) applied

(5) Queue attribute (wait_queue)

Description: Define the queue attribute regarding how tasks are queued waiting for the semaphore.
Definition format: Symbol
Definition range: Select either of the following:

- TA_TFIFO : Queued in FIFO order
- TA_TPRI : Queued in order of task priority

When omitting: The set value in default cfg file (factory setting: TA_TFIFO) applied

8.4.9 Eventflag Definition (flag[])

The definition item flag[] is provided for the definition (creation) of eventflags.

❑Format

```
flag[(1) ID number]{
        name              = (2) ID name;
        initial_pattern   = (3) Initial bit pattern of eventflag;
        wait_queue        = (4) Queue attribute;
        wait_multi        = (5) Multiple wait permission attribute;
        clear_attribute   = (6) Clear attribute;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Initial bit pattern of eventflag (initial_pattern)

Description: Define the initial bit pattern of the eventflag.
Definition format: Numeric value
Definition range: 0 - 0xFFFFFFFF
When omitting: The set value in default cfg file (factory setting: 0) applied

(4) Queue attribute (wait_queue)

Description: Define the queue attribute regarding how tasks are queued waiting for the eventflag.
Note that if TA_WSGL is specified for wait_multi, this definition item has no effect. Also, even in the case where TA_WMUL is specified for wait_multi, if clear_attribute is chosen to be NO, the queue is managed in FIFO order regardless of how this definition item is specified.
Definition format: Symbol
Definition range: Select either of the following:

- TA_TFIFO : Queued in FIFO order
- TA_TPRI : Queued in order of task priority

When omitting: The set value in default cfg file (factory setting: TA_TFIFO) applied

(5) Multiple wait permission attribute (wait_multi)

Description: Define the attribute regarding whether multiple tasks are permitted to wait for the eventflag.
Definition format: Symbol
Definition range: Select either of the following:

- TA_WMUL : Multiple tasks are permitted to wait
- TA_WSGL : Multiple tasks not permitted to wait

When omitting: The set value in default cfg file (factory setting: TA_WSGL) applied

(6) Clear attribute (clear_attribute)

Description: Define the clear attribute (TA_CLR) of the eventflag.
Definition format: Symbol
Definition range: Select either of the following:

- YES : Clear attribute set
- NO : Clear attribute not set

When omitting: The set value in default cfg file (factory setting: NO) applied

8.4.10 Data Queue Definition (dataqueue[])

The definition item dataqueue[] is provided for the definition (creation) of data queues.

❑Format

```
dataqueue[(1) ID number]{
        name             = (2) ID name;
        buffer_size      = (3) Maximum data count of the data queue;
        wait_queue       = (4) Queue attribute;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Maximum data count of the data queue (buffer_size)

Description: Define the maximum number of data in the data queue. The size of data queue area is buffer_size × 4 (bytes).
A data queue with data count = 0 can be created.
Definition format: Numeric value
Definition range: 0 - 65535
When omitting: The set value in default cfg file (factory setting: 0) applied

(4) Queue attribute (wait_queue)

Description: Define the queue attribute regarding how tasks are queued waiting to send. Note that the receive queue is always managed in FIFO order.
Definition format: Symbol
Definition range: Select either of the following:

- TA_TFIFO : Queued in FIFO order
- TA_TPRI : Queued in order of task priority

When omitting: The set value in default cfg file (factory setting: TA_TFIFO) applied

8.4.11 Mailbox Definition (mailbox[])

The definition item mailbox[] is provided for the definition (creation) of mailboxes.

❑Format

```
mailbox[(1) ID number]{
        name              = (2) ID name;
        wait_queue        = (3) Queue attribute;
        message_queue     = (4) Message queue attribute;
        max_pri           = (5) Maximum priority of messages;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Queue attribute (wait_queue)

Description: Define the queue attribute regarding how tasks are queued waiting for the mailbox.
Definition format: Symbol
Definition range: Select either of the following:

- TA_TFIFO : Queued in FIFO order
- TA_TPRI : Queued in order of task priority

When omitting: The set value in default cfg file (factory setting: TA_TFIFO) applied

(4) Message queue attribute (message_queue)

Description: Define the manner in which messages are tied to the message queue.
Definition format: Symbol
Definition range: Select either of the following:

- TA_MFIFO : Message queue in FIFO order
- TA_MPRI : Message queue in order of message priority

When omitting: The set value in default cfg file (factory setting: TA_MFIFO) applied

(5) Maximum priority of messages (max_pri)

Description: If TA_MPRI is specified for message_queue, define the maximum priority of messages here.
Definition range: 1 to system.message_pri
When omitting: The set value in default cfg file (factory setting: 1) applied
Remark: If message_queue = TA_MFIFO, this definition item has no effect.

8.4.12 Mutex Definition (mutex[])

The definition item mutex[] is provided for the definition (creation) of mutexes

❑Format

```
mutex[(1) ID number]{
        name             = (2) ID name;
        ceilpri          = (3) Ceiling priority;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Ceiling priority (ceilpri)

Description: The mutexes in the RI600/PX are controlled by a priority ceiling protocol. Define this ceiling priority here.
Definition format: Numeric value
Definition range: 1 - system.priority
When omitting: The set value in default cfg file (factory setting: 1) applied

8.4.13 Message Buffer Definition (message_buffer[])

The definition item message_buffer[] is provided for the definition (creation) of message buffers.

❑Format

```
message_buffer[(1) ID number]{
        name                  = (2) ID name;
        mbf_size              = (3) Size of message buffer;
        mbf_section           = (4) Section name assigned to the message buffer area;
        max_msgsz             = (5) Maximum message size;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Size of message buffer (mbf_size)

Description: Specify the size of the message buffer in bytes. A message buffer which size is 0 can be created. In this case, the transmit and receive sides of the message buffer are fully synchronized when they communicate.
Definition format: Numeric value
Definition range: 0 or a multiple of 4 in the range 8 to 65,532
When omitting: The set value in default cfg file (factory setting: 0) applied

(4) Section name assigned to the message buffer area (mbf_section)

Description: Define the section name to be assigned to the message buffer. The section attribute is "DATA", and the alignment number is 4.
When linking, be sure to allocate this section in the RAM area.
Usually, this section should not be in a memory object since only the kernel accesses to message buffer area.
And this section must not be allocated to address 0.
Definition format: Symbol
Definition range: -
When omitting: The set value in default cfg file (factory setting: BRI_RAM) applied
Remark: If mbf_size = 0, this definition item has no effect.

(5) Maximum message size (max_msgsz)

Description: Define the maximum message size in bytes. The specified value is rounded up to a multiple of 4. If mbf_size > 0, max_msgsz must be equal to or less than (mbf_size - 4).
Definition format: Numeric value
Definition range: 1 - 65528
When omitting: The set value in default cfg file (factory setting: 4) applied

8.4.14 Fixed-sized Memory Pool (memorypool[])

The definition item memorypool[] is provided for the definition (creation) of fixed-sized memory pools.

❑Format

```
memorypool[(1) ID number]{
        name            = (2) ID name;
        section         = (3) Section name assigned to the pool area (section);
        num_block       = (4) Number of memory blocks;
        siz_block       = (5) Memory block size;
        wait_queue      = (6)Queue attribute;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Section name assigned to the pool area (section)

Description: Define the section name to be assigned to the pool area. The section attribute is "DATA", and the alignment number is 4.
When linking, be sure to allocate this section in the RAM area.
This section should be in a memory object to enable access from tasks that acquire memory blocks. And this section must not be allocated to address 0.
Definition format: Symbol
Definition range: -
When omitting: The set value in default cfg file (factory setting: BURI_HEAP) applied

(4) Number of memory blocks (num_block)

Description: Define the number of blocks in the memory pool.
Note that the size of the memory pool area is determined by num_block ✕ siz_block (bytes). The upper limit of the pool size is VTMAX_AREASIZE (bytes).
Definition format: Numeric value
Definition range: 1 - 65535
When omitting: The set value in default cfg file (factory setting: 1) applied

(5) Memory block size (siz_block)

Description: Define the memory block size in bytes.
Definition format: Numeric value
Definition range: 4 - 65535
When omitting: The set value in default cfg file (factory setting: 256) applied

(6) Queue attribute (wait_queue)

Description: Define the queue attribute regarding how tasks are queued waiting for the memory block.
Definition format: Symbol
Definition range: Select either of the following:

- TA_TFIFO : Queued in FIFO order
- TA_TPRI : Queued in order of task priority

When omitting: The set value in default cfg file (factory setting: TA_TFIFO) applied

8.4.15 Variable-sized Memory Pool Definition (variable_memorypool[])

The definition item variable_memorypool[] is provided for the definition (creation) of variable-sized memory pools.

❑Format

```
variable_memorypool[(1) ID number]{
        name                 = (2) ID name;
        mpl_section          = (3) Section name assigned to the pool area (section);
        heap_size            = (4) Size of memory pool;
        max_memsize          = (5) Upper limit of the memory block size;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Section name assigned to the pool area (mpl_section)

Description: Define the section name to be assigned to the pool area. The section attribute is "DATA", and the alignment number is 4.
When linking, be sure to allocate this section in the RAM area.
This section should be in a memory object to enable access from tasks that acquire memory blocks. And this section must not be allocated to address 0.
Definition format: Symbol
Definition range: -
When omitting: The set value in default cfg file (factory setting: BURI_HEAP) applied

(4) Size of memory pool (heap_size)

Description: Define the size of the memory pool in bytes. The specified value is rounded up to a multiple of 4.
Definition format: Numeric value
Definition range: 24 to VTMAX_AREASIZE
When omitting: The set value in default cfg file (factory setting: 1024) applied

(5) Upper limit of the memory block size (max_memsize)

Description: Define the upper limit of an acquirable memory block size in bytes.
The maximum size that can be actually acquired might become larger than max_memsize.
Definition format: Numeric value
Definition range: 1 to 0xBFFFFF4 (192MB - 12)
When omitting: The set value in default cfg file (factory setting: 36) applied
Supplementation: In the current implementation of the kernel, the size of memory block actually acquired is selected from 12 kinds of variation in the maximum. This variation is decided according to max_memsize. Table 8.9shows variation of memory block size. Note, this behavior will be subjected to change in the future.

Table 8.9 Variation of Memory Block Size

No.	Memory block size (in hexadecimal)	Example-1 : max_memsize == 0x100	Example-2 : max_memsize == 0x20000
1	12 (0xC)	Used	-
2	36 (0x24)	Used	-
3	84 (0x54)	Used	Used
4	180 (0xB4)	Used	Used
5	372 (0x174)	-	Used
6	756 (0x2F4)	-	Used
7	1524 (0x5F4)	-	Used
8	3060 (0xBF4)	-	Used
9	6132 (0x17F4)	-	Used
10	12276 (0x2FF4)	-	Used
11	24564 (0x5FF4)	-	Used
12	49140 (0xBFF4)	-	Used
13	98292 (0x17FF4)	-	Used
14	196596 (0x2FFF4)	-	Used
15	393204 (0x5FFF4)	-	-
16	786420 (0xBFFF4)	-	-
17	1572852 (0x17FFF4)	-	-
18	3145716 (0x2FFFF4)	-	-
19	6291444 (0x5FFFF4)	-	-
20	12582900 (0xBFFFF4)	-	-
21	25165812 (0x17FFFF4)	-	-
22	50331636 (0x2FFFFF4)	-	-
23	100663284 (0x5FFFFF4)	-	-
24	201326580 (0xBFFFFF4)	-	-

8.4.16 Cyclic Handler Definition (cyclic_hand[])

The definition item cyclic_hand[] is provided for the definition (creation) of cyclic handlers.

❑Format

```
cyclic_hand[(1) ID number]{
        name                = (2) ID name;
        entry_address       = (3) Handler entry address;
        interval_counter    = (4) Activation cycle;
        start               = (5) Operating state of the cyclic handler;
        phsatr              = (6) Preservation of the activation phase;
        phs_counter         = (7) Activation phase;
        exinf               = (8) Extended information;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Handler entry address (entry_address)

Description: Define the entry address of the cyclic handler starting from which it is executed.
Definition format: Function name
Definition range: -
When omitting: Cannot be omitted (error assumed)

(4) Activation cycle (interval_counter)

Description: Define a cycle time in ms at which intervals the handler is activated.
Definition range: 1 - (0x7FFFFFFF - system.tic_nume) / system.tic_deno
When omitting: The set value in default cfg file (factory setting: 1) applied

(5) Operating state of the cyclic handler (start)

Description: Define the attribute regarding the handler's operating state.
Definition format: Symbol
Definition range: Select either of the following:

- ON : Cyclic handler placed in an active state (TA_STA attribute specified)
- OFF : Cyclic handler not placed in an active state (TA_STA attribute not specified)

When omitting: The set value in default cfg file (factory setting: OFF) applied

(6) Preservation of the activation phase (phsatr)

Description: Define the handler attribute regarding whether its activation phase is saved.
Definition format: Symbol
Definition range: Select either of the following:

- ON : Activation phase saved (TA_PHS attribute specified)
- OFF : Activation phase not saved (TA_PHS attribute not specified)

When omitting: The set value in default cfg file (factory setting: OFF) applied

(7) Activation phase (phs_counter)

Description: Define the handler activation phase in ms. The activation phase must be equal to or less than the activation cycle.
Definition format: Numeric value
Definition range: 0 to interval_counter
When omitting: The set value in default cfg file (factory setting: 0) applied

(8) Extended information (exinf)

Description: Define the extended information of the cyclic handler.
Definition format: Numeric value
Definition range: 0 to 0xFFFFFFFF
When omitting: The set value in default cfg file (factory setting: 0) applied

8.4.17 Alarm Handler Definition (alarm_hand[])

The definition item alarm_hand[] is provided for the definition (creation) of alarm handlers.

❑Format

```
alarm_hand[(1) ID number]{
        name              = (2) ID name;
        entry_address     = (3) Handler entry address;
        exinf             = (4) Extended information;
};
```

❑Contents

(1) ID number

Description: Define the ID number.
Definition format: Numeric value
Definition range: 1 - 255
When omitting: The ID number is assigned automatically.

(2) ID name (name)

Description: Define the ID name. The specified ID name is output to the ID name header file (kernel_id.h) in the form given below.

```
#define  <ID name>  <ID number>
```

Definition format: Symbol
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Handler entry address (entry_address)

Description: Define the entry address of the alarm handler starting from which it is executed.
Definition format: Function name
Definition range: -
When omitting: Cannot be omitted (error assumed)

(4) Extended information (exinf)

Description: Define the extended information of the alarm handler.
Definition format: Numeric value
Definition range: 0 - 0xFFFFFFFF
When omitting: The set value in default cfg file (factory setting: 0) applied

8.4.18 Relocatable Vector Definition (interrupt_vector[])

The definition item interrupt_vector[] is used to define interrupt handlers for relocatable vectors.

When any interrupt occurs whose vector number is not defined here, the system goes down.

Note that the cfg600px does not generate code to initialize the interrupt control registers (e.g., IPL), the causes of interrupts, etc. for the interrupts defined here. These initialization routines need to be created in the startup file or in any way deemed appropriate for the application developed by the user.

❑Attention

Since the vector number from 1 to 8 are reserved by the kernel, do not define interrupt handlers for these vector. And do not define the vectors which are reserved by the MCU specificatio.

❑Format

```
interrupt_vector[(1) Vector number]{
        entry_address       = (2) Handler entry address;
        os_int              = (3) Kernel interrupt specification;
        pragma_switch       = (4) Switch passed to PRAGMA extension function;
};
```

❑Contents

(1) Vector number

Description: Define the vector number of the interrupt that defines the handler.
Definition format: Numeric value
Definition range: 0 - 255
When omitting: Cannot be omitted (error assumed)

(2) Handler entry address (entry_address)

Description: Define the entry address of the interrupt handler starting from which it is executed.
Definition format: Function name
Definition range: -
When omitting: Cannot be omitted (error assumed)

(3) Kernel interrupt specification (os_int)

Description: Define whether the interrupt defined here is a kernel interrupt.
Interrupts whose priority level is lower than or equal to the kernel interrupt mask level (system.system_IPL) must be handled as "kernel interrupt", and the other interrupts must be handled as "non-kernel interrupt".
Note, all relocatable interrupts must be handled as "kernel interrupt" when system.system_IPL is 15.
Definition format: Symbol
Definition range: Select either of the following:

- YES : Kernel interrupt
- NO : Non-kernel interrupt

When omitting: Cannot be omitted (error assumed)

(4) Switch passed to PRAGMA extension function (pragma_switch))

Description: For the function specified by entry_address, the cfg600px outputs a #pragma interrupt directive to kernel_id.h as interrupt function. Specify the switch to be passed to this pragma directive. For details about program specifications, see the compiler's manual.

Definition format: Symbol

Definition range: One of the following can be specified. To specify multiple choices, separate each with a comma. Note, both ACC and NOACC cannot be specified at the same time.

- E: The "enable" switch that permits a multiple interrupt is passed.
- F: The "fint" switch that specifies a fast interrupt is passed. Note, a fast interrupt must be handled as non-kernel interrupt (os_int=NO).
- S: The "save" switch that limits the number of registers used by the interrupt handler is passed.
- ACC : The "acc" switch that the interrupt handler guarantees the ACC register is passed.
- NOACC : The "noacc" switch that the interrupt handler does not guarantees the ACC register is passed.

Table 8.10 shows treatment of the ACC register.

Table 8.10 Treatment of the ACC Register

"ACC", "NOACC"	Compiler "save_acc" option	
	without "save_acc"	with "save_acc"
None	Both "acc" and "noacc" switch is not passed. The ACC register is not guaranteed.	Both "acc" and "noacc" switch is not passed. The ACC register is guaranteed.
"ACC"	"acc" switch is passed. The ACC register is guaranteed.	
"NOACC"	"noacc" switch is passed. The ACC register is not guaranteed.	

When omitting: No switches are passed.

8.4.19 Fixed Vector Definition (interrupt_fvector[])

The definition item interrupt_fvector[] is used to define the interrupt handlers for fixed vectors. The causes of fixed-vector interrupts are all handled as non-kernel interrupts.

Although the causes of fixed-vector interrupts in microcomputer specifications are not assigned vector numbers, the vector addresses in the RI600/PX each are assigned a vector number, as shown in Table 8.11. Table 8.11 also shows behavior when an vector is not defined.

Table 8.11 Fixed vector number

Vector Address	Vector Number	Factor *1	Behavior when an vector is not defined
0xFFFFFF80	0	Endian select register	The following is set according to "endian" option for compiler. endian=little : 0xFFFFFFFF endian=big : 0xFFFFFFF8
0x FFFFFF84	1	(Reserved)	0xFFFFFFFF
0x FFFFFF88	2	Option function select register 1	
0x FFFFFF8C	3	Option function select register 0	
0x FFFFFF90	4	(Reserved)	
0x FFFFFF94	5	(Reserved)	
0x FFFFFF98	6	(Reserved)	
0x FFFFFF9C	7	ROM code protection (Flash memory)	
0x FFFFFFA0	8	ID code protection on connection of the on-chip debugger (Flash memory)	
0x FFFFFFA4	9		
0x FFFFFFA8	10		
0x FFFFFFAC	11		
0x FFFFFFB0	12	(Reserved)	
0x FFFFFFB4	13	(Reserved)	
0x FFFFFFB8	14	(Reserved)	
0x FFFFFFBC	15	(Reserved)	
0x FFFFFFC0	16	(Reserved)	System down
0x FFFFFFC4	17	(Reserved)	
0x FFFFFFC8	18	(Reserved)	
0x FFFFFFCC	19	(Reserved)	
0x FFFFFFD0	20	Privileged instruction exception	
0x FFFFFFD4	21	Access exception	Access exception handler
0x FFFFFFD8	22	(Reserved)	System down
0x FFFFFFDC	23	Undefined instruction exception	
0x FFFFFFE0	24	(Reserved)	
0x FFFFFFE4	25	Floating-point exception	
0x FFFFFFE8	26	(Reserved)	
0x FFFFFFEC	27	(Reserved)	
0x FFFFFFF0	28	(Reserved)	
0x FFFFFFF4	29	(Reserved)	
0x FFFFFFF8	30	Non-maskable interrupt	
0x FFFFFFFC	31	Reset	PowerON_Reset_PC()

Note: The factors are different according to MCU.

Note that the cfg600px does not generate code to initialize the interrupt control registers (e.g., IPL), the causes of interrupts, etc. for the interrupts defined here. These initialization routines need to be created in the startup file or in any way deemed appropriate for the application developed by the user.

❑Attention

Please do not define interrupt handlers for MCU's reserved vector.

And do not define a handler to the vector-21. If defined, the access exception handler (_RI_sys_access_exception()) never be initiated.

❑Format

```
interrupt_fvector[(1) Vector number]{
        entry_address      = (2) Handler entry address;
        pragma_switch      = (3) Switch passed to PRAGMA extension function;
};
```

❑Contents

(1) Vector number

Description: Define the vector number of each interrupt referring to Table 8.11.
Definition format: Numeric value
Definition range: 0 - 31
When omitting: Cannot be omitted (error assumed)

(2) Handler entry address (entry_address)

Description: Define the entry address of the interrupt handler starting from which it is executed, or value for the fixed vector.
Definition format: Function name or numeric value
Definition range: 0 - 0xFFFFFFFF, when numeric value is specified
When omitting: Cannot be omitted (error assumed)

(3) Switch passed to PRAGMA extension function (pragma_switch))

Description: For the function specified by entry_address, the cfg600px outputs a #pragma interrupt directive to kernel_id.h as interrupt function. Specify the switch to be passed to this pragma directive. For details about program specifications, see the compiler's manual.

Note,

Note, #pragma interrupt is not generated, when entry_address is specified by numeric value or the vector number is 31 (reset).

Definition format: Symbol
Definition range: One of the following can be specified. To specify multiple choices, separate each with a comma. Note, both ACC and NOACC cannot be specified at the same time.

- S: The "save" switch that limits the number of registers used by the interrupt handler is passed.
- ACC : The "acc" switch that the interrupt handler guarantees the ACC register is passed.
- NOACC : The "noacc" switch that the interrupt handler does not guarantees the ACC register is passed.

Table 8.10 shows treatment of the ACC register.
When omitting: No switches are passed.

8.5 Executing the Configurator

8.5.1 Outline of the Configurator

The configurator is the tool that based on the content defined in the cfg file, outputs various system definition files and the header files used for the application. Following files are output by executing the configurator.

- ID number header file (kernel_id.h)
  This file defines the ID numbers of kernel objects.
- Service call definition file (kernel_sysint.h)
  This is the declaration file needed to invoke service calls using the INT instruction. This file is included from kernel.h.
- System definition files (kernel_rom.h, kernel_ram.h, ri600.inc)
  The file kernel_rom.h and kernel_ram.h must be included by startup file. Please refer to Section 7.2, "Creating Startup File (resetprg.c)."
  The file ri600.inc is included from the ritable.src that is generated by mkritblpx.
- Vector table template file (vector.tpl)
  The vector.tpl is loaded into mkritblpx.
- CMT definition file (ri_cmt.h)
  When one of CMT0, CMT1, CMT2 or CMT3 is specified for clock.timer, the file indicated by clock.template retrieved from the directory defined by environment variable "LIB600", and the file is copied and renamed to "ri_cmt.h".
  The ri_cmt.h is included by startup file.

The outline operation of the configurator is shown in Figure 8.2.

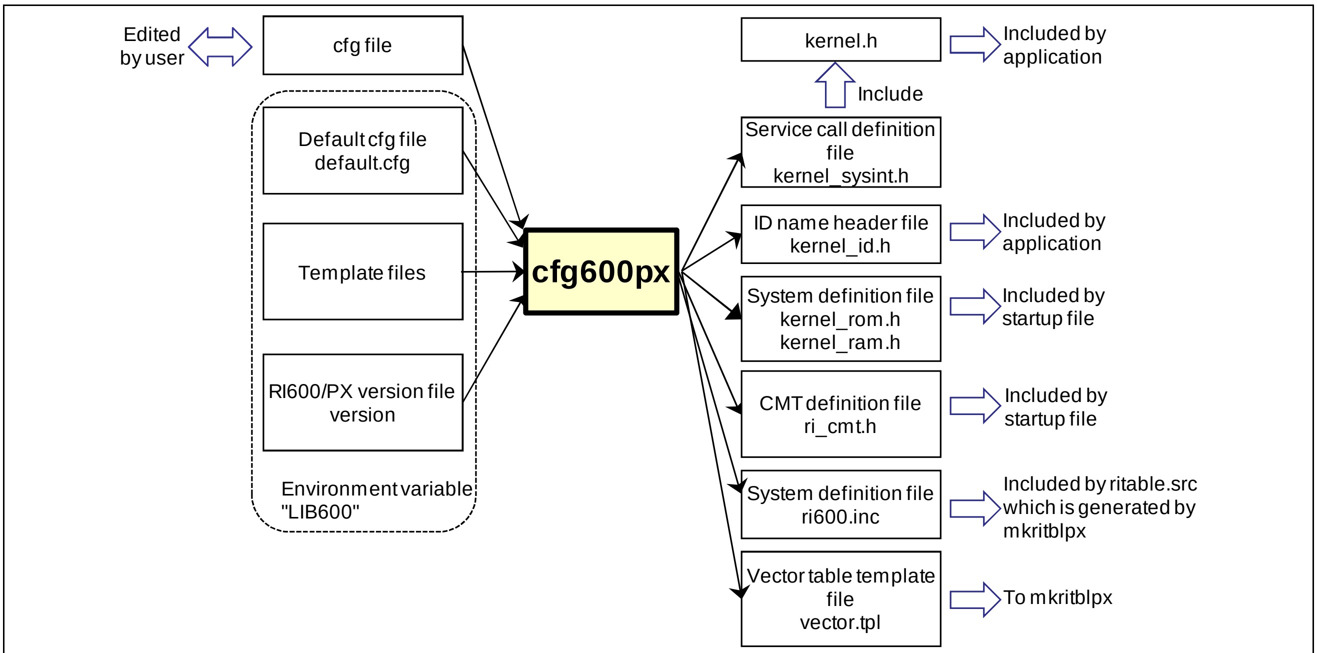

Figure 8.2 Outline Operation of the Configurator

8.5.2 Environment Settings

Following environment variables need to be set.

- LIB600
  "<Installation directory>\lib600"

8.5.3 Configurator Start Procedure

The configurator is started in the form shown below.

```
cfg600px[Δ<option>…][Δ<cfg file name>]
```

If the filename extension of the cfg file is omitted, the extension ".cfg" is assumed.

8.5.4 Command Options

(1) -U option

When an undefined interrupt occurs, a system-down results.When -U option is specified, the vector number will be transferred to the system-down routine (Refer to "6.8 System-Down Routine".). This is useful for debugging. However, the kernel code size increases by about 1.5 kB.

(2) -v option

Shows a description of the command option and details of its version.

(3) -V option

Shows the creation status of files generated by the command.

8.6 Errors Messages

8.6.1 Error Output Format and Error Levels

This section describes the meaning of the error messages output in the following format.

```
Error number (Error level) Error message
```

Errors are classified into two levels as shown in Table 8.12.

Table 8.12 Error Levels

Error level		Operation
(W)	Warning	Continues processing.
(E)	Error	Aborts processing.

8.6.2 List of Messages

(1) Error Messages

01001 (E) Syntax error.

Syntax error.

01002 (E) Illegal XXX --> <setting>

(1) The setting for definition name XXX is illegal.

(2) The number of XXX definitions exceeds the threshold limit value.

01003 (E) Unknown token --> <XXX>

The specified strings cannot be recognized as a definition name.

01005 (E) Task[1]'s priority is too large. --> <setting>

task[].priority exceeds system.priority.

01006 (E) clock.IPL is too large.--> <setting>

clock.IPL exceeds system.system_IPL.

01007 (E) System timer's vector <XXX> conflict

A interrupt_vector[] has already defined for the CMT's vector when one of CMT0, CMT1, CMT2, or CMT3 is specified for clock.timer.

01009 (E) XXX is already defined.

XXX has already been defined.

01010 (E) XXX[YYY] is already defined.

XXX[YYY] has already been defined.

01013 (E) Zero divide error

Zero division expression.

01015 (E) Can't specify F switch when os_int=YES.

"F" switch cannot be specified for kernel interrupt handler.

01016 (E) interrupt_vector[YYY].os_int must be YES.

Relocatable interrupt cannot be used as kernel interrupt when the kernel interrupt mask level (system.system_IPL) is 15.

01018 (E) mailbox[YYY].max_pri(setting) is bigger than system.message_pri(setting).

mailbox.max_pri must be lower than or equal to system.message_pri.

01019 (E) Neither system.tic_nume nor system.tic_deno is 1.

Neither system.tic_nume nor system.tic_deno is 1.

01020 (E) Symbols other than NO and NO are defined simultaneously.

Symbols other than NO and NO are defined simultaneously.

01022 (E) semaphore[YYY].initial_count is bigger than semaphore[YYY].max_count semaphore[YYY].initial_count is bigger than semaphore[YYY].max_count.

`O1023 (E) Size of memorypool[YYY] is larger than VTMAX_AREASIZE`

The size of fixed-sized memory pool must be lower than or equal to VTMAX_AREASIZE (256MB).

`O1024 (E) variable_memorypool[YYY].max_memsize is larger than 192MB-12`

variable_memorypool.max_memsize must be lower than or equal to "192MB-12".

`O1025 (E) mutex[YYY].ceilpri is bigger than system.priority.`

mutex.ceilpri must be lower than or equal to system.priority.

`O1026 (E) XXX is not a multiple of 4.`

XXX must be multiple of 4.

`O1027 (E) max_msgsz (setting) is larger than mbf_size(setting) - 4 .`

message_buffer.max_msgsz must be lower than or equal to message_buffer.mbf_size-4.

`O1028 (E) variable_memorypool[YYY].max_memsize is too large. (Max.=ZZZ)`

The value YYY specified for variable_memorypool[YYY].max_memsize is too large. The maximum value of max_memsize that can be specified for heap_size is ZZZ.

`O1029 (E) Timer tick is too long.`

Time-tick cycle decided by system.tic_nume and system.tic_deno is too long. Shorten timer-tick or lower clock.timer_clock.

`O1030 (E) Timer tick is too short.`

Time-tick cycle decided by system.tic_nume and system.tic_deno is too short. Lengthen timer-tick or higher clock.timer_clock.

`O1033 (E) start_address must be smaller than end_address.`

The start_address must be less smaller the end_address.

`O1034 (E) ACC and NOACC can't be specified at the same time.`

ACC and NOACC can not be specified at the same time.

`O1034 (E) task[XXX].domain_num(=YYY) is bigger than maxdefine.max_domain(=ZZZ).`

The task[XXX].domain_num setting (YYY) exceeds maxdefine.max_domain setting (ZZZ).

`O2001 (E) Not enough memory`

Insufficient memory.

`O2003 (E) Illegal argument --> <XXX>`

The startup format has an error.

`O2004 (E) Can't write open <file name>`

The file cannot be generated.

`O2005 (E) Can't open <file name>`

The file cannot be accessed in the directory indicated by environment variable "LIB600".

`O2006 (E) Can't open version file`

The version file cannot be found in the current directory or the directory indicated by environment variable "LIB600".

`O2007 (E) Can't open default configuration file`

The default.cfg file cannot be found in the current directory or the directory indicated by environment variable "LIB600".

`O2008 (E) Can't open configuration file <file name>.`

The specified cfg file cannot be accessed.

`O2009 (E) XXX is not defined`

XXX is not defined.

`O2010 (E) Initial start task is not defined.`

There is no task[] definition with initial_start=ON.

`O2011 (E) Environment variable "LIB600" not prepared.`

Environment variable "LIB600" is not set.

`O2013 (E) memory_object is not defined`

There is no memory_object[] definition.

(2) Warning Messages

`O3001 (W) XXX is already defined.`

XXX has already been defined. The definition is ignored.

`O3002 (W) The maximum ID of the object is larger than maxdefine.xxx.`

The maximum object ID ex extended since the maxdefine.XXX is less than the ID number specified by the object definition or the number of the object definitions.

`O4001 (W) XXX is not defined.`

The definition of XXX is omitted; the setting in the default cfg file is used.

`O4005 (W) Timer counter value is less than your setting time`

The specified timer cycle (= system.tic_nume / system.tic_deno [ms]) cannot be achieved without the error margin. Actual time of cycle is shorter than the specified time of the cycle.

9. Table Generation Utility (mkritblpx)

9.1 Summary

The utility mkritblpx is a command line tool that after collecting service call information used in the application, generates service call tables and interrupt vector tables.

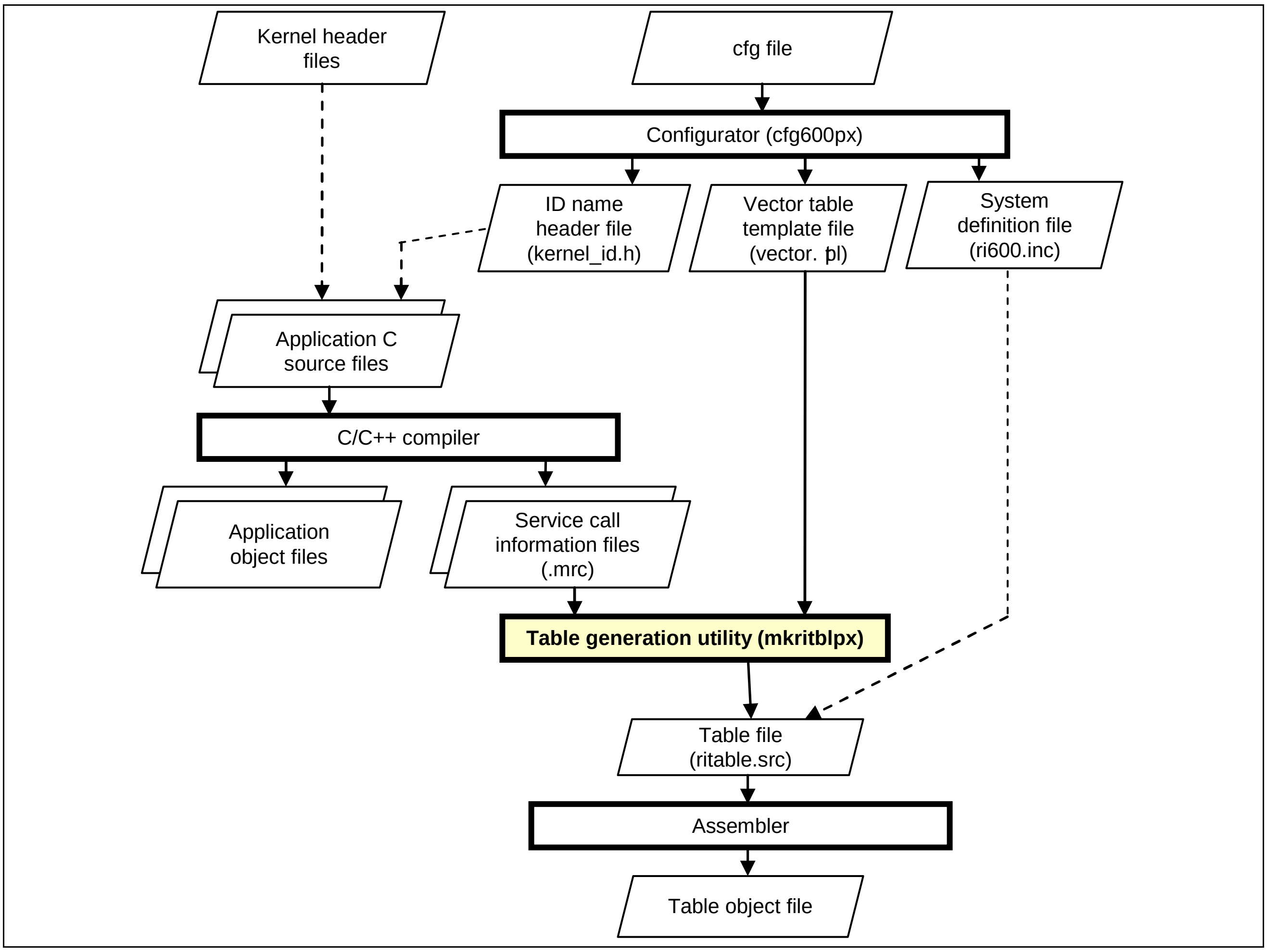

Figure 9.1 Outline of mkritblpx

In kernel_sysint.h that is included by kernel.h, it is so defined that when service call functions are used, the service call information will be output to the .mrc file by the .assert control instruction. Using these service call information files as its input, mkritblpx generates a service call table in such a way that only the service calls used in the system will be linked.

Furthermore, mkritblpx generates an interrupt vector table based on the vector table template files output by cfg600px and the .mrc file.

9.2 Environment Setup

Following environment variables need to be set.

- LIB600
  "<Installation directory>\lib600"

9.3 Table Generation Utility Start Procedure

The table generation utility is started in the form shown below.

```
C:\> mkritblpx  <directory name or file name>
```

For the parameter, normally specify the directory that contains the mrc file that is generated when compiled. Multiple directories or files can be specified.

Note that the mrc file present in the current directory is unconditionally selected for input.

Also, it is necessary that vector.tpl generated by cfg600px be present in the current directory.

9.4 Notes

Please specify mrc files generated by compilation of application without omission. If some service call modules might not be build into the load module when there is an omission in the specification of mrc files, When unlinked service call is issued, the system will go down.

10. Sample Program

The High-performance Embedded Workshop can generates a RI600/PX project. This chapter explains "RI600/PX Project" generated by High-performance Embedded Workshop.

10.1 Summary of Sample Program

There are three domains, "Master domain", "Application domain-A" and "Application domain-B".

The master domain (domid #1) is "trusted domain". The master domain creates various objects that are required to execute application domain-A and -B. The task that belongs to the master domain (MasterDom_Task) is created and activated by the cfg file.

The application domain-A (domid #2) and -B (domid #3) are not "trusted domain".

The task that belongs to the application domain-A is AppDomA_Task, and the task that belongs to the application domain-B is AppDomB_Task.

AppDomA_Task and AppDomB_Task access the global variable "g_ulSharedData" by using the semaphore (ID_SEM1) while controlling it exclusively.

And AppDomA_Task sends data to the data queue (ID_DTQ1), AppDomB_Task receives it.

Table 10.1 Kernel Objects

Type	ID number, etc.	Description
Domain	1	• Master domain • Trusted domain • Belonging task : "MasterDom_Task"
	2	• Application domain-A • Untrusted domain • Belonging task : "AppDomA_Task"
	3	• Application domain-B • Untrusted domain • Belonging task : "AppDomB_Task"
Task	ID_MASTERDOMTASK	Created and activated by the cfg file
	ID_DOM_A_TASK	Created and activated by MasterDom_Task
	ID_DOM_B_TASK	Created and activated by MasterDom_Task
Semaphore	ID_SEM1	• Created by MasterDom_Task • Control to access to variable "g_ulSharedData" from AppDomA_Task and AppDomB_Task
Data queue	ID_DTQ1	• Created by MasterDom_Task • Used to communicate between AppDomA_Task and AppDomB_Task • The data queue area is generated in the "BS" section. This section is outside of memory objects.
Variable-sized memory pool	ID_MPL1	• Created by MasterDom_Task • It is dummy. • The pool area is generated in the "BU_SH" section. This section is inside of memory_object[4].
Cyclic handler	ID_CYC1	• Created and activated by the cfg file • Rotate ready queue for AppDomA_Task and AppDomB_Task
Alarm handler	ID_ALM1	• Created by the cfg file • It is dummy.
Interrupt handler	Relocatable vector #64	• Defined by the cfg file • It is dummy.

10.2 Generating a RI600/PX Project

1. Select the Create a new project workspace option from the Welcome! dialog box and click the OK button or select [File -> New Workspace]. The New Project Workspace dialog box will be displayed.
2. Enter the name of the new workspace into the Workspace Name field.
3. Select "RX" as the [CPU family] and "Renesas RX Standard" as the "Tool chain".
4. Select "Application" as [Project type] and click the [OK] button to create the new workspace and project. This then launches the wizard you have selected to guide you through the creation process.
5. Select "RI600/PX" as the [RTOS] in the [New Project -2/11- Select RTOS].

10.3 Generated Files

Here, it explains the main files.

(1) resetprg.c

Refer to section 7.2,"Creating Startup File (resetprg.c)."

(2) <Project name>.cfg

This is the cfg file.

Please modify "clock.template" setting according to the MCU to be used.

(3) <Project name>.c

The tasks and handlers described in Table 10.1 are implemented.

10.4 Memory Map

The “aligned_section” linker option, which aligns the start address of the section to 16-bytes boundary, is specified for the sections indicated by parentheses "[ ]".

10.4.1 RAM Area

Table 10.2 RAM Area

Address	Section Order (setting for linker)	Description	Memory Object
0 - 0x0001FFFF	SI	System stack	Non memory object
	BRI_RAM,RRI_RAM	Kernel data	
	BS,BS_1,BS_2,RS,RS_1,RS_2	Data only for handlers	
	[SURI_STACK]	User stack	
	[BU_MASTERDOM],BU_MASTERDOM_1, BU_MASTERDOM_2, RU_MASTERDOM,RU_MASTERDOM_1, RU_MASTERDOM_2	Data only for the master domain	memory_object[1]
	[BU_DOM_A],BU_DOM_A_1,BU_DOM_A_2, RU_DOM_A,RU_DOM_A_1,RU_DOM_A_2	Data only for the application domain-A	memory_object[2]
	[BU_DOM_B],BU_DOM_B_1,BU_DOM_B_2, RU_DOM_B,RU_DOM_B_1,RU_DOM_B_2	Data only for the application domain-B	memory_object[3]
	[BURI_HEAP],BU_SH,BU_SH_1,BU_SH_2, RU_SH,RU_SH_1,RU_SH_2	Shared data	memory_object[4]

10.4.2 ROM Area

Table 10.3 ROM Area

Address	Section Order (setting for linker)	Description	Memory Object
0xFFFF0000 - 0xFFFFFF7F	[PU_MASTERDOM],CU_MASTERDOM, CU_MASTERDOM_1,CU_MASTERDOM_2, DU_MASTERDOM, DU_MASTERDOM_1,DU_MASTERDOM_2	Code and constant only for the master domain	memory_object[5]
	[PU_DOM_A],CU_DOM_A,CU_DOM_A_1, CU_DOM_A_2,DU_DOM_A,DU_DOM_A_1, DU_DOM_A_2	Code and constant only for the application domain-A	memory_object[6]
	[PU_DOM_B],CU_DOM_B,CU_DOM_B_1, CU_DOM_B_2,DU_DOM_B,DU_DOM_B_1, DU_DOM_B_2	Code and constant only for the application domain-B	memory_object[7]
	[PU_SH],WU_SH,WU_SH_1,WU_SH_2,LU_SH, CU_SH,CU_SH_1,CU_SH_2, DU_SH,DU_SH_1,DU_SH_2	Shared code and data	memory_object[8]
	INTERRUPT_VECTOR	Relocatable vector table	Non memory object
	PRI_KERNEL	Kernel code	
	CRI_ROM,DRI_ROM	Kernel constant	
	C$*,PS,CS,CS_1,CS_2,DS,DS_1,DS_2	Code and constant only for handlers	
0xFFFFFF80 - 0xFFFFFFFF	FIX_INTERRUPT_VECTOR	Fixed vector table	

10.4.3 Memory Objects

There are eight memory objects, these are registered in the cfg file.

Memory object must be started from 16-bytes boundary address. Therefore, it is required to specify "aligned_section "option for the start section of memory objects (memory_object[].start_address) at linking.

```
// Master domain data
memory_object[1]{
    start_address = BU_MASTERDOM;
    end_address   = RU_MASTERDOM_2;
    acptn1        = 0x0001;          Only the master domain can read.
    acptn2        = 0x0001;          Only the master domain can write.
    acptn3        = 0;               No domain can execute.
};
```

Figure 10.1 memory_object[1] : Data only for the Master Domain

```
// App-domain A data
memory_object[2]{
    start_address = BU_DOM_A;
    end_address   = RU_DOM_A_2;
    acptn1        = 0x0002;          Only the application domain-A can read.
    acptn2        = 0x0002;          Only the application domain-A can write.
    acptn3        = 0;               No domain can execute.
};
```

Figure 10.2 memory_object[2] : Data only for the Application Domain-A

```
// App-domain B data
memory_object[3]{
    start_address = BU_DOM_B;
    end_address   = RU_DOM_B_2;
    acptn1        = 0x0004;          Only the application domain-B can read.
    acptn2        = 0x0004;          Only the application domain-B can write.
    acptn3        = 0;               No domain can execute.
};
```

Figure 10.3 memory_object[3] : Data only for the Application Domain-B

```
// Shared data
memory_object[4]{
    start_address = BURI_HEAP;
    end_address   = RU_SH_2;
    acptn1        = TACP_SHARED;     All domains can read.
    acptn2        = TACP_SHARED;     All domains can write.
    acptn3        = 0;               No domain can execute.
};
```

Figure 10.4 memory_object[4] : Shared Data

```
// Master domain code and const
memory_object[5]{
    start_address = PU_MASTERDOM;
    end_address   = DU_MASTERDOM_2;
    acptn1        = 0x0001;              Only the master domain can read.
    acptn2        = 0;                   No domain can write.
    acptn3        = 0x0001;              Only the master domain can execute.
};
```

Figure 10.5 memory_object[5] : Code and Constant only for the Master Domain

```
// App-domain A code and const
memory_object[6]{
    start_address = PU_DOM_A;
    end_address   = DU_DOM_A_2;
    acptn1        = 0x0002;              Only the application domain-A can read.
    acptn2        = 0;                   No domain can write.
    acptn3        = 0x0002;              Only the application domain-A can execute.
};
```

Figure 10.6 memory_object[6] : Code and Constant only for the Application Domain-A

```
// App-domain B code and const
memory_object[7]{
    start_address = PU_DOM_B;
    end_address   = DU_DOM_B_2;
    acptn1        = 0x0004;              Only the application domain-B can read.
    acptn2        = 0;                   No domain can write.
    acptn3        = 0x0004;              Only the application domain-B can execute.
};
```

Figure 10.7 memory_object[7] : Code and Constant only for the Application Domain-B

```
// Shared code and const
memory_object[8]{
    start_address = PU_SH;
    end_address   = DU_SH_2;
    acptn1        = TACP_SHARED;         All domains can read.
    acptn2        = 0;                   No domain can write.
    acptn3        = TACP_SHARED;         All domains can execute.
};
```

Figure 10.8 memory_object[8] : Shared Code and Constant

10.4.4 User Stacks

The user stacks must be allocated to the outside of memory objects. In this sample, user stacks for all tasks are generated in SURI_STACK section that is the default setting.

(1) User Stack for MasterDom_Task

MasterDom_Task is created statically by the cfg file.

```
task[]{
    name             = ID_MASTERDOMTASK;
    entry_address    = MasterDom_Task();
    initial_start    = ON;
    stack_size       = 256;
    priority         = 1;
//  stack_section    = SURI_STACK;            The user stack is generated in the
    exinf            = 1;                     SURI_STACK section when
    domain_num       = 1;                     "stack_section" is omitted.
};
```

Figure 10.9 Creation of MasterDom_Task

(2) User Stacks for AppDomA_Task and AppDomB_Task

AppDomA_Task and AppDomB_Task are generated by acre_tsk which is called by MasterDom_Task. The start address and size of user stacks for each tasks are passed to acre_tsk.

User stack area for both AppDomA_Task and AppDomB_Task are generated in SURI_STACK section by using #pragma section directive.

```
/****************************************************************************
 Stack for AppDomA_Task and AppDomB_Task
****************************************************************************/
#define DOM_A_STKSZ     0x100          // AppDomA_Task's stack size
#define DOM_B_STKSZ     0x100          // AppDomB_Task's stack size
#pragma section B SURI_STACK
static UW       s_ulDomA_Stk[DOM_A_STKSZ];      // Stack area for AppDomA_Task
static UW       s_ulDomB_Stk[DOM_B_STKSZ];      // Stack area for AppDomB_Task
```

Figure 10.10 User Stacks for AppDomA_Task and AppDomB_Task

10.5 Setting of Build Tools concerning Sections

10.5.1 Standard Library Generator

The section of standard library is made memory objects that can be accessed from all domains.

Table 10.4 Sections of Standard Library

Area	Section	Memory object
Code	PU_SH	memory_object[8]
Constant	CU_SH	
Literal	LU_SH	
Switch branch table	WU_SH,WU_SH_1,WU_SH2_	
Initialized data	DU_SH,DU_SH_1,DU_SH_2	
Uninitialized data	BU,BU_SH_1,BU_SH_2	memory_object[4]
Initialized data(RAM) (specify for linker)	RU_SH,RU_SH_1,RU_SH_2	

10.5.2 C/C++ Compiler

The default section is same as Table 10.4. When other than this section is required, the section is changed by using "#pragma section" directive.

10.5.3 Linker

The "aligned_section" option is required for the following sections.

1. The section specified for "memory_object[].start_address"
2. The section specified for "task[].stack_section" (In this sample, this is not specified.)
3. SURI_STACK

Therefore, in this sample, the "aligned_section" is specified for the start section of memory objects and SURI_STACK.

10.6 Example of Dealing with Access Violation

This sample implements following example. For details, refer to sample code.

- Raise task exception, and the task exception handling routine re-activates itself.
- Do processing over again from normal point by using longjump().

11. Stack Size Estimation

11.1 Types of Stacks

There are two types of stacks: the system stack and the user stack. The method for calculating stack sizes differs between the system stack and user stack.

- User stack
  The stacks for tasks are referred to as the user stack. The user stack is accessed by the CPU's USP register.
  The user stack size must be multiple of 16, and the start address of the user stack area must be 16-bytes boundary. If not, an error is detected. Table 11.1 shows how to specify user stack.

Table 11.1 How to specify user stack

Item	Static creation ("task[] in the cfg file)	Dynamic creation (cre_tsk, acre_tsk service call)
User stack size	The user stack size is specified by "task[].stack_size". When the specified value is other than multiple of 16, the cfg600px reports an error.	The user stack size is specified by T_CTSK.stksz. When the specified value is other than multiple of 16,cre_tsk and acre_tsk return an error.
User stack area	The cfg600px generates the user stack area in the section specified by "task[].stack_section". When the start address of the user stack area, vsta_knl detects an error and enters to system-down.	The area from the address specified by T_CTSK.stk with the size specified by T_CTSK.stksz is used for the user stack area. This area should be secured by application. When the start address of the user stack area is not 16-bytes boundary, cre_tsk and acre_tsk return an error.

- System stack
  There is only one stack in the system, provided for other than tasks, that is used in common by various handlers and the kernel.
  The system stack size is specified by system.stack_size in the cfg file. The section name is "SI".
  The system stack is accessed by the CPU's ISP register.

11.2 "Call Walker"

The compiler package includes "Call Walker" which is a utility tool to calculate stack size.

The Call Walker can display stack size used by each function tree.

11.3 User Stack Size Estimation

The quantity consumed of user stack for each task is a value in which the value calculated by the following expressions was rounded up to the multiple of 16.

$$\text{Quantity consumed of user stack} = treesz\_task + ctxsz\_task + treesz\_tex + ctxsz\_tex$$

treesz_task:

Size consumed by function tree that makes the task entry function starting point. (The size displayed by Call Walker)

treesz_tex:

Size consumed by function tree that makes the task exception handling routine entry function starting point. (The size displayed by Call Walker). When task exception handling routine is not used, *treesz_tex* is assumed to be 0.

ctxsz_task, *ctxsz_tex*:

Task context size.
The *ctxsz_task* is for task and the *ctxsz_tex* is for task exception handling routine. When task exception handling routine is not used, *ctxsz_tex* is assumed to be 0.
Task context size is different according to system.context defined in the cfg file. See Table 11.2.

Table 11.2 Task context size

system.context	Task context size (bytes)
NO	68
FPSW	72
ACC	76
FPSW,ACC	80
MIN	44
MIN,FPSW	48
MIN,ACC	52
MIN,FPSW,ACC	56

11.4 System Stack Size Estimation

The system stack is most often consumed when an interrupt occurs during service call processing followed by the occurrence of multiple interrupts.[6] The quantity consumed of system stack is calculated by the following expressions.

$$\text{Quantity consumed of system stack} = svcsz + \left(\sum_{k=1}^{15} treesz\_inthdr_k\right) + sysdwnsz$$

svcsz:

The maximum size among the service calls to be used in the system. The value *svcsz* depends on the kernel version. See release notes.

$treesz\_inthdr_k$:

Size consumed by function tree that makes the interrupt handler entry function starting point. (The size displayed by Call Walker)
The "k" is interrupt priority level. If there are multiple interrupts in the same priority level, the $treesz\_inthdr_k$ should select the maximum size among the handlers.
The size used by the system clock interrupt handler (the interrupt priority level is specified by cloclk.IPL in the cfg file) is the following maximum values. Please refer to the release notes for *clocksz1*, *clocksz*2 and *clocksz*3.
Don't have to add the size used by the system clock interrupt hanlder to the system stack size when system timer is not used (clock.timer = NOTIMER).

- *clocksz*1 + *cycsz*
- *clocksz*2 + *almlsz*
- *clocksz*3
  - ♦ *cycsz*
    Size consumed by function tree that makes the cyclic handler entry function starting point. (The size displayed by Call Walker)
    If there are multiple cyclic handlers, the *cycsz* should select the maximum size among the handlers
  - ♦ *almsz*
    Size consumed by function tree that makes the alarm handler entry function starting point. (The size displayed by Call Walker)
    If there are multiple alarm handlers, the *almsz* should select the maximum size among the handlers

sysdwnsz:

Size consumed by function tree that makes the system down rourtine entry function starting point. (the size displayed by Call Walker) + 40. When the system down routine has never been executed, *sysdwnsz* is assumed to be 0.

[6] After switchover from user stack to system stack

RI600/PX V.1.00 User's Manual

Publication Date: Rev.1.00 September 1, 2011

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.1

RI600/PX V.1.00

Renesas Electronics

R20UT0606EJ0100