



T-Engine Forum

AMP T-Kernel Standard Extension Specification

Ver. 1.00.00

TEF021-S003-01.00.00/en

March 2009

T-Engine Forum

<http://www.t-engine.org/>

AMP T-Kernel Standard Extension Specification (Ver.1.00.00)

TEF021-S003-01.00.00/en

March 2009

Copyright © 2006-2009 T-Engine Forum. All Rights Reserved.

T-Engine Forum owns the copyright of this specification.

Permission of T-Engine Forum is necessary for copying, republishing, posting on servers, or redistribution to lists of the contents of this specification.

The contents written in this specification may be changed without a prior notice for improvement or other reasons in the future.

About this specification, please refer to follows;

Publisher

T-Engine Forum

The 28th Kowa Building 2-20-1 Nishi-gotanda
Shinagawa-Ward Tokyo 141-0031 Japan

TEL : +81-3-5437-0572 FAX : +81-3-5437-2399

E-mail : office@t-engine.org

Contents

| | |
|---|-----------|
| 1.AMP T-Kernel Standard Extention Overview | 13 |
| 1.1 Overview of AMP T-Kernel Standard Extension | 13 |
| 1.2 Available Functions | 14 |
| 1.3 Target Operating Environment | 14 |
| 2.Concepts Underlying the AMP TKSE..... | 16 |
| 2.1 Processor and Kernel..... | 16 |
| 2.1.1 Processor | 16 |
| 2.1.2 Kernel | 16 |
| 2.1.3 Synchronization and Communication Between Kernels..... | 16 |
| 2.2 Process..... | 16 |
| 2.2.1 Definition of Process..... | 16 |
| 2.2.2 Address Space of Process..... | 18 |
| 2.2.3 Process State and Task State | 19 |
| 2.2.4 Process/Task Priority and Scheduling..... | 21 |
| 2.2.5 Execution Environment of Process | 22 |
| 2.2.6 User Process and System Process..... | 23 |
| 2.2.7 Creating a Process | 23 |
| 2.2.8 Combination with AMP T-Kernel Programs..... | 25 |
| 2.3 Synchronization and Communication..... | 27 |
| 2.3.1 Interprocess Synchronization and Communication | 27 |
| 2.3.2 Intertask Synchronization and Communication..... | 28 |
| 2.3.3 Intertask Synchronization and Communication Function..... | 28 |
| 2.4 Object Management..... | 29 |
| 2.4.1 Retrieving Object ID Number..... | 29 |
| 2.5 Standard File Management and Standard Input/Output Functions | 31 |
| 2.5.1 File Management of AMP TKSE | 31 |
| 2.5.2 Standard File Management Function | 32 |
| 2.5.3 Standard Input/Output Function..... | 32 |
| 2.5.4 Shared Filesystem Management | 32 |
| 2.6 Device Management and Event Management Functions | 34 |
| 2.6.1 Access to T-Kernel Device and Event Notification | 34 |
| 2.6.2 Device Management Function | 34 |
| 2.6.3 Event Management Function..... | 34 |
| 3.AMP T-Kernel Standard Extension Common Specifications..... | 36 |
| 3.1 Data Types | 36 |
| 3.1.1 Basic Data Types | 36 |
| 3.1.2 Other Defined Data Types | 37 |
| 3.2 Error Codes | 38 |
| 3.2.1 Overview..... | 38 |
| 3.2.2 List of Error Codes | 39 |

| | |
|---|------------|
| 4.AMP T-Kernel Standard Extension Functions | 43 |
| 4.1 Memory Management Function | 43 |
| 4.1.1 Overview of the Memory Management Function | 43 |
| 4.1.2 System Calls | 44 |
| 4.1.3 Library Calls | 49 |
| 4.2 Process/Task Management Function | 57 |
| 4.2.1 Process/Task Management Function Overview | 57 |
| 4.2.2 System Calls | 58 |
| 4.3 Interprocess Message Function | 87 |
| 4.3.1 Interprocess Message Function Overview | 87 |
| 4.3.2 Message Type | 88 |
| 4.3.3 Message Structure | 90 |
| 4.3.4 System Message | 90 |
| 4.3.5 Message Handler | 91 |
| 4.3.6 System Calls | 94 |
| 4.4 Global Name | 104 |
| 4.4.1 Overview of the Global Name | 104 |
| 4.4.2 System Calls | 105 |
| 4.5 Synchronous Intertask Communication Function | 109 |
| 4.5.1 Synchronous Intertask Communication Function Overview | 109 |
| 4.5.2 System Calls (Semaphore) | 110 |
| 4.5.3 System Calls (Mutex) | 116 |
| 4.5.4 System Calls (Eventflag) | 122 |
| 4.5.5 System Calls (Mailbox) | 130 |
| 4.5.6 System Calls (Message buffer) | 136 |
| 4.5.7 System Calls (Rendezvous Port) | 142 |
| 4.6 Object Management | 150 |
| 4.6.1 Overview | 150 |
| 4.6.2 System Calls | 151 |
| 4.7 Standard Input/Output Function | 155 |
| 4.7.1 Standard Input/Output Function Overview | 155 |
| 4.7.2 Target File System | 156 |
| 4.7.3 File Access | 157 |
| 4.7.4 Initial State of File Descriptor | 157 |
| 4.7.5 Disk Cache | 157 |
| 4.7.6 File Name | 158 |
| 4.7.7 Path Name | 160 |
| 4.7.8 Root Directory | 160 |
| 4.7.9 Current Directory | 161 |
| 4.7.10 This Directory "." and Parent Directory ".." | 161 |
| 4.7.11 Error Code | 162 |
| 4.7.12 System Calls | 163 |

| | | |
|-------------|---|------------|
| 4.8 | Standard File Management Function | 203 |
| 4.8.1 | Standard File Management Function Overview | 203 |
| 4.8.2 | File and Link..... | 203 |
| 4.8.3 | File System..... | 205 |
| 4.8.4 | Connect File System | 205 |
| 4.8.5 | File ID..... | 206 |
| 4.8.6 | Link | 206 |
| 4.8.7 | Working File | 207 |
| 4.8.8 | Path Name | 207 |
| 4.8.9 | File Type | 209 |
| 4.8.10 | Normal File Composition | 209 |
| 4.8.11 | Record Number / Current Record | 209 |
| 4.8.12 | Link File Composition | 210 |
| 4.8.13 | File Control..... | 211 |
| 4.8.14 | Reference Count of File | 211 |
| 4.8.15 | File Access | 212 |
| 4.8.16 | File System Management Information | 213 |
| 4.8.17 | File Management Information..... | 214 |
| 4.8.18 | Link Structure | 216 |
| 4.8.19 | System Calls | 218 |
| 4.9 | Filesystem Sharing Management | 280 |
| 4.9.1 | Filesystem Sharing Management Overview | 280 |
| 4.9.2 | System Call | 281 |
| 4.10 | Event Management | 283 |
| 4.10.1 | Event Management Overview | 283 |
| 4.10.2 | Event Type..... | 284 |
| 4.10.3 | Event Creation from Device Event Notifications | 285 |
| 4.10.4 | Priority of Event Queue and Event..... | 287 |
| 4.10.5 | Keyboard Events | 288 |
| 4.10.6 | Character Code..... | 288 |
| 4.10.7 | Pointing Device Event..... | 289 |
| 4.10.8 | Designates the Operation Type of the Pointing Device..... | 289 |
| 4.10.9 | Wheel Support | 289 |
| 4.10.10 | Event Structure | 290 |
| 4.10.11 | System Calls | 293 |
| 4.11 | Device Management Function | 308 |
| 4.11.1 | Device Management Function Overview | 308 |
| 4.11.2 | Basic Concepts..... | 308 |
| 4.11.3 | System Calls | 311 |
| 4.12 | Time Management | 324 |
| 4.12.1 | Time Management Overview | 324 |
| 4.12.2 | System Calls | 326 |

| | | |
|--------|---|-----|
| 4.12.3 | Library Calls | 329 |
| 4.13 | System Management Function | 334 |
| 4.13.1 | System Management Function Overview | 334 |
| 4.13.2 | System Calls | 335 |
| 4.14 | Shared Library Function | 339 |
| 4.14.1 | Shared Library Function Overview | 339 |
| 4.14.2 | Library Call..... | 340 |
| 5. | Implementation Method | 344 |
| 5.1 | Overview | 344 |
| 5.2 | Memory Management and Segment Management | 344 |
| 5.3 | Process/Task Management | 345 |
| 5.4 | Interprocess Messages | 347 |
| 5.5 | Intertask Synchronization and Communication Functions | 348 |
| 5.6 | Device Management Function | 348 |
| 5.7 | Time Management Function..... | 348 |
| 5.8 | Object Management..... | 348 |
| 6. | Configuration..... | 350 |
| 6.1 | System Configuration Information | 350 |

List of Figures

| | |
|--|-----|
| [Figure 1] System Configuration of AMP TKSE..... | 13 |
| [Figure 2] Relationship between processes and main task/subtasks | 17 |
| [Figure 3] Local memory space and shared memory space..... | 19 |
| [Figure 4] Task State Transitions | 21 |
| [Figure 5] Position of File Management | 31 |
| [Figure 6] Shared File Management..... | 33 |
| [Figure 7] Positions of Device Management and Event Management..... | 34 |
| [Figure 8] Process state | 66 |
| [Figure 9] Relation of file and links | 204 |
| [Figure 10] Example of the order of appearance in pathname | 207 |
| [Figure 11] Change of the record number by record deletion | 210 |
| [Figure 12] Position of Event Management | 283 |
| [Figure 13] KeyMap structure..... | 303 |

Index of API

| | | |
|--------------|--|----------|
| calloc | Allocate Nonresident Local Memory | 50 |
| DATEtoTIME | Convert Calendar Date to total number of seconds | 330 |
| dladdr | Retrieve Symbol Information of Shared Library | 343 |
| dlclose | Close Shared Library | 342 |
| dlopen | Open Shared Library | 340 |
| dlsym | Find Symbol of Shared Library | 341 |
| free | Free Nonresident Local Memory | 52 |
| GMTtoLT | local time compensation | 332 |
| LTtoGMT | standard time compensation | 333 |
| malloc | Allocate Nonresident Local Memory | 49 |
| realloc | Reallocate Nonresident Local Memory | 51 |
| Scalloc | Allocate Nonresident Common Memory | 54 |
| Sfree | Free Nonresident Common Memory | 56 |
| Smalloc | Allocate Nonresident Common Memory | 53 |
| Srealloc | Reallocate Nonresident Common Memory | 55 |
| TIMEtoDATE | Convert consecutive seconds the total number of seconds to Calendar Date | 331 |
| tkse_acp_por | Accept Rendezvous | 146 |
| tkse_apd_rec | Append Record | 243 |
| tkse_att_fls | Attach File system | 265 |
| tkse_attach | Attach file system | 151, 163 |
| tkse_brk_msg | Notify The Occurrence of Event | 101 |
| tkse_cal_por | Call Rendezvous | 145 |
| tkse_can_tmg | Cancel Time-out Message | 100 |
| tkse_can_wup | Cancel Task Wake-up Request | 81 |
| tkse_chdir | Modify Current Directory | 188 |
| tkse_chg_emk | Change System Event Mask | 298 |
| tkse_chg_fat | Change File Access Attribute | 254 |
| tkse_chg_fls | Change File system Information | 270 |
| tkse_chg_fmd | Change File Access Mode | 252 |
| tkse_chg_fnm | Change File name | 256 |
| tkse_chg_fsm | Change File system Connection Mode | 278 |
| tkse_chg_ftm | Change File Date and Time | 257 |
| tkse_chg_pri | Change Priority of Processes/Tasks | 63 |
| tkse_chg_wrk | Change Working File | 220 |
| tkse_chk_fil | Check File Access Privileges | 250 |
| tkse_chmod | Change File Mode | 190 |
| tkse_close | Close file / directory | 168 |
| tkse_clr_evt | Clear Event | 295 |
| tkse_clr_flg | Clear Eventflag | 126 |
| tkse_clr_msg | Clear Message | 98 |

| | | |
|----------------|--|----------|
| tkse_cls_dev | Close Device | 313 |
| tkse_cls_fil | Close File | 229 |
| tkse_cre_fil | Create File | 221 |
| tkse_cre_flg | Create Eventflag | 122 |
| tkse_cre_lnk | Create Link File | 223 |
| tkse_cre_mbf | Create Messagebuffer | 136 |
| tkse_cre_mbx | Create Mailbox | 130 |
| tkse_cre_mtx | Create Mutex | 116 |
| tkse_cre_nam | Create Global Name Data | 105 |
| tkse_cre_por | Create Rendezvous Port | 142 |
| tkse_cre_prc | Create/Execute Processes | 58 |
| tkse_cre_sem | Create Semaphore | 110 |
| tkse_cre_tsk | Subtask Creation | 74 |
| tkse_creat | Create File | 193 |
| tkse_crs_tsk | Subtask Creation and Startup | 76 |
| tkse_def_msg | Define Message Handler | 102 |
| tkse_del_fil | Delete File | 230 |
| tkse_del_flg | Delete Eventflag | 124 |
| tkse_del_mbf | Delete Messagebuffer | 138 |
| tkse_del_mbx | Remove Mailbox | 132 |
| tkse_del_mtx | Delete Mutex | 118 |
| tkse_del_nam | Remove Global Name Data | 107 |
| tkse_del_por | Delete Rendezvous Port | 144 |
| tkse_del_rec | Delete Record | 245 |
| tkse_del_sem | Delete Semaphore | 112 |
| tkse_det_fls | Detach File system | 267 |
| tkse_detach | Detach file system | 152, 164 |
| tkse_dly_tsk | Delay Task | 82 |
| tkse_dup | Replicate File Descriptor | 185 |
| tkse_dup2 | Replicate File Descriptor | 186 |
| tkse_ext_prc | Exit Process | 61 |
| tkse_ext_tsk | Exit invoking task | 77 |
| tkse_fchdir | Modify Current Directory | 189 |
| tkse_fchmod | Change File Mode | 192 |
| tkse_fil_sts | Get file information | 258 |
| tkse_fls_sts | Get File system Management Information | 269 |
| tkse_fnd_lnk | Find Link Record | 235 |
| tkse_fnd_rec | Find Record | 233 |
| tkse_fstat | Get file information | 180 |
| tkse_fsync | File's Disk Cache Content and Disk Synchronization | 187 |
| tkse_ftruncate | Set File Size to the Specified Length | 198 |
| tkse_fwd_por | Forward Rendezvous | 147 |

| | | |
|----------------|---|----------|
| tkse_gen_fil | Generate File Directly | 225 |
| tkse_get_dev | Retrieve Device Name | 320 |
| tkse_get_dfm | Get default access mode | 263 |
| tkse_get_etm | Get Event Timer Value | 297 |
| tkse_get_evt | Get Event | 293 |
| tkse_get_inf | Get Statistics Information about Processes | 67 |
| tkse_get_krm | Get Auto Repeat Target Key | 304 |
| tkse_get_krp | Get Auto Repeat Interval | 306 |
| tkse_get_lnk | Get Link to File | 218 |
| tkse_get_mbk | Get Memory Block | 44 |
| tkse_get_nam | Get Global Name Data | 108 |
| tkse_get_nlk | Get Links Sequentially | 272 |
| tkse_get_otm | Refer System Uptime | 328 |
| tkse_get_pdp | Get PD Position | 307 |
| tkse_get_tid | Get Invoking Task ID | 83 |
| tkse_get_tim | Refer System Time | 326 |
| tkse_get_ver | Get version | 338 |
| tkse_getdents | Get directory entry | 172 |
| tkse_getfsstat | Retrieve a List of File systems | 200 |
| tkse_getlink | Retrieve a LINK to Standard File | 202 |
| tkse_ins_rec | Insert Record | 241 |
| tkse_las_evt | Get Elapsed Time from the Last Event Occurrence | 301 |
| tkse_lnk_sts | Get Link File Information | 260 |
| tkse_loc_mtx | Lock Mutex | 119 |
| tkse_loc_rec | Record lock | 248 |
| tkse_lod_mod | Load Load Module | 84 |
| tkse_lod_spg | Load System Programs | 335 |
| tkse_lseek | Move the current position of a file/directory | 169 |
| tkse_lst_dev | Retrieve Registered Devices | 323 |
| tkse_lst_fls | Get File system | 273 |
| tkse_lstat | Get file information | 179 |
| tkse_map_rec | Map Record | 275 |
| tkse_mbk_sts | Refer to Memory State | 47 |
| tkse_mkdir | Make directory | 183 |
| tkse_ofl_sts | Get file information | 259 |
| tkse_open | Open File/Directory | 154, 165 |
| tkse_opn_dev | Open Device | 311 |
| tkse_opn_fil | Open File | 227 |
| tkse_oref_dev | Retrieve Device Information | 322 |
| tkse_prc_inf | Retrieve Various Information about Processes | 71 |
| tkse_prc_sts | Get Process State | 65 |
| tkse_put_evt | Event Occurrence | 294 |

| | | |
|---------------|--|-----|
| tkse_rcv_mbf | Receive from Messagebuffer | 140 |
| tkse_rcv_mbx | Receive from Mailbox | 134 |
| tkse_rcv_msg | Receive Message | 96 |
| tkse_rea_dev | Read Device Data (Asynchronous) | 314 |
| tkse_rea_rec | Read Record | 237 |
| tkse_read | Read file | 170 |
| tkse_ref_dev | Retrieve Device Information | 321 |
| tkse_ref_flg | Refer to Eventflag State | 129 |
| tkse_ref_mbf | Refer to Message Buffer State | 141 |
| tkse_ref_mbx | Refer to Mailbox State | 135 |
| tkse_ref_mtx | Refer to Mutex State | 121 |
| tkse_ref_por | Refer to Rendezvous Port State | 149 |
| tkse_ref_sem | Refer to Semaphore State | 115 |
| tkse_rel_mbk | Release Memory Block | 46 |
| tkse_removefs | Removing Shared Filesystem | 282 |
| tkse_rename | Rename file | 181 |
| tkse_req_emg | Process Exit Message | 69 |
| tkse_req_evt | Request Event Message | 299 |
| tkse_req_tmng | Request Time-out Message | 99 |
| tkse_ret_msg | Exit Message Handler | 103 |
| tkse_rmdir | Remove directory | 184 |
| tkse_rpl_rdv | Reply to Rendezvous | 148 |
| tkse_see_rec | Move Current Record | 232 |
| tkse_set_dfm | Set default access mode | 264 |
| tkse_set_flg | Set Eventflag | 125 |
| tkse_set_krm | Set Auto Repeat Target Key | 302 |
| tkse_set_krp | Set Auto Repeat Interval | 305 |
| tkse_set_tim | Set System Time | 327 |
| tkse_sharefs | Sharing of Filesystems | 281 |
| tkse_sig_sem | Return Semaphore Resource | 113 |
| tkse_slp_tsk | Task Sleep | 79 |
| tkse_snd_mbf | Send to Messagebuffer | 139 |
| tkse_snd_mbx | Send to Mailbox | 133 |
| tkse_snd_msg | Send Message | 94 |
| tkse_srea_dev | Read Device Data (Synchronous) | 315 |
| tkse_sta_tsk | Subtask Startup | 75 |
| tkse_stat | Get file information | 174 |
| tkse_sus_dev | Request to Suspend Device | 319 |
| tkse_swri_dev | Write Data to Device (Synchronous) | 317 |
| tkse_syn_fil | Synchronize on File Basis | 279 |
| tkse_syn_fls | Synchronize File system | 268 |
| tkse_syn_lnk | Synchronize Link File | 261 |

| | | |
|---------------|---|-----|
| tkse_sync | Disk Cache Content and Disk Synchronization | 199 |
| tkse_ter_prc | Terminate Other Process | 62 |
| tkse_ter_tsk | Terminate Other Task | 78 |
| tkse_trc_rec | Truncate Record Size | 246 |
| tkse_truncate | Set File Size to the Specified Length | 197 |
| tkse_umask | Set File Creation Mask | 195 |
| tkse_ump_rec | Unmap Record | 277 |
| tkse_unl_mod | Unload Load Module | 86 |
| tkse_unl_mtx | Unlock Mutex | 120 |
| tkse_unl_spg | Unload System Programs | 337 |
| tkse_unlink | Unlink directory entry | 182 |
| tkse_utimes | Modify Access Time, Modification Time | 194 |
| tkse_wai_dev | Wait for Request Completion for Device | 318 |
| tkse_wai_flg | Wait Eventflag | 127 |
| tkse_wai_sem | Get Semaphore Resource | 114 |
| tkse_wri_dev | Write Data to Device (Asynchronous) | 316 |
| tkse_wri_rec | Write Record | 239 |
| tkse_write | Write file | 171 |
| tkse_wup_tsk | Task Wake up | 80 |
| tkse_xch_fil | Exchange File Content | 247 |

1. AMP T-Kernel Standard Extension Overview

1.1 Overview of AMP T-Kernel Standard Extension

AMP T-Kernel Standard Extension (hereinafter called AMP TKSE) is a functional extension program for AMP T-Kernel.

A program that extends the T-Kernel functions to realize more advanced OS functions is called T-Kernel Extension. Standard Extension refers to standard-specification T-Kernel Extension, which adds T-Kernel to functions generally required for large systems such as file management and process management.

AMP T-Kernel is a real-time Operating System (OS) that is an expansion of T-Kernel to support Asymmetric Multiple Processors (AMP).

AMP TKSE is a T-Kernel Extension that is an expansion of Standard Extension to support AMP T-Kernel.

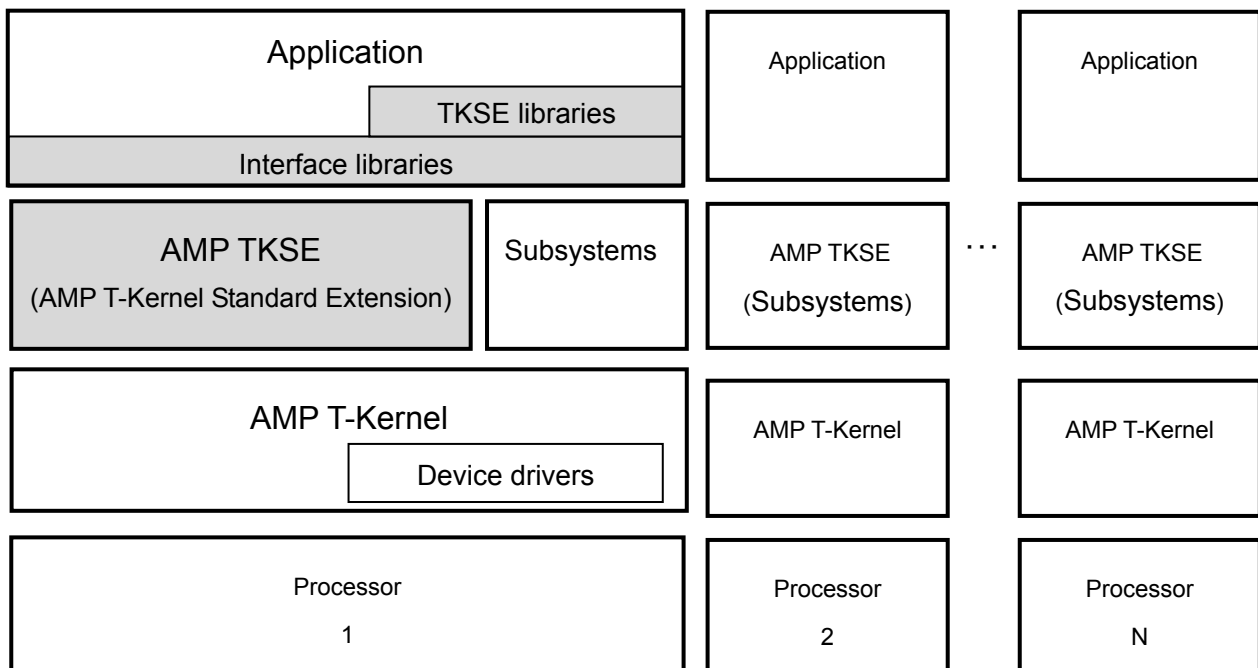
T-Kernel, Standard Extension, and AMP T-Kernel refer to the following versions in this specification.

T-Kernel version 1.00

T-Kernel AMP TKSE version 1.00

AMP T-Kernel version 1.00

A general system configuration for AMP TKSE is shown below.



[Figure 1] System Configuration of AMP TKSE

AMP consists of multiple processors. One copy of AMP T-Kernel operates on each processor.

AMP TKSE is a T-Kernel Extension that operates on AMP T-Kernel. The main features are implemented as a subsystem of AMP T-Kernel. The library to call extended SVC for these subsystems in C language function is

called an interface library. Part of the AMP TKSE functions is provided as TKSE libraries that directly link to applications, not subsystems.

A user-created program that runs on AMP TKSE is called an application. An application uses various functions using the APIs (Application Programming Interfaces) of AMP TKSE. Among the AMP TKSE APIs, functions to be invoked using interface libraries are called system calls, and functions to be invoked using TKSE libraries are called library function calls.

In AMP, AMP T-Kernel, AMP TKSE, and the application exist in each processor. The application can synchronize and communicate with other applications using the functions provided by AMP TKSE.

1.2 Available Functions

AMP TKSE provides the following functions:

- Memory management
- Process/Task management
- Interprocess message
- Global name
- Intertask synchronization and communication
- Standard input/output
- Standard file management
- Shared File System Management
- Event management
- Device management
- Time management
- System management
- Shared library

The details of above functions and API specifications are explained in later chapters.

1.3 Target Operating Environment

All the functions of AMP T-Kernel need to be available in an environment in which AMP TKSE is to run. Moreover, AMP T-Kernel alone can operate in environments where neither the MMU (Memory Management Unit) of CPU nor shared memory between processors exists while MMU and shared memory between processors are essential for AMP TKSE.

The operation of AMP TKSE requires the following T-Kernel device drivers that conform to the T-Engine Standard Device Driver Specifications:

- System disk driver: Used for memory management, process/task management, and standard file management, etc.

AMP TKSE also uses the following device drivers. However, these device drivers are not required if such functions are not to be used.

- Clock driver: Gets and sets the RTC date/time in time management.
- KB/PD driver: Receives KB/PD events in event management.
- Console driver: Inputs/Outputs console in standard.

If any of the device drivers listed above is dependent on other drivers and/or subsystems (such as the PCMCIA card manager), these drivers and/or subsystems are also required.

In AMP TKSE, a memory space is accessed as a logical space using MMUs. It is required, therefore, that the device driver can normally access the buffer area of a logical space allocated by a process. The driver must be capable of switching between task spaces, making a space resident, and converting a logical address to a physical address.

2. Concepts Underlying the AMP TKSE

2.1 Processor and Kernel

2.1.1 Processor

The hardware of AMP consists of multiple processors. A processor is identified by the processor ID number. The processor ID number is the ID number defined by AMP T-Kernel.

One AMP T-Kernel operates on each processor, and a copy of AMP TKSE operates on each AMP T-Kernel. Applications operate on a processor of their own. Applications do not usually get operated by specifying the processor directly.

2.1.2 Kernel

In AMP, an AMP T-Kernel exists on each processor. In addition, it is conceivable that multiple AMP T-Kernels and SMP T-Kernel may exist in mixed system of SMP and AMP in the future. AMP T-Kernel and SMP T-Kernel are also simply called, "Kernel".

One AMP TKSE exists in and supports each AMP T-Kernel one-to-one. In other words, in AMP, all processors, AMP T-Kernels, AMP TKSEs, and applications associated with each other in one-to-one fashion. Applications usually do not need to be aware of the processor directly, and only must be aware of levels higher than the kernel.

2.1.3 Synchronization and Communication Between Kernels

Applications in AMP TKSE can synchronize and communicate with applications of other AMP TKSEs within the same AMP systems. This is called synchronization and communication between kernels. Synchronization and communication between kernels are realized with various operations for other AMP TKSE programs and resources using the following functions provided by AMP TKSE.

- (1) Interprocess messages
- (2) Synchronization and communication between tasks
- (3) File system sharing

Details and API specifications for each function above will be described in another chapter.

2.2 Process

2.2.1 Definition of Process

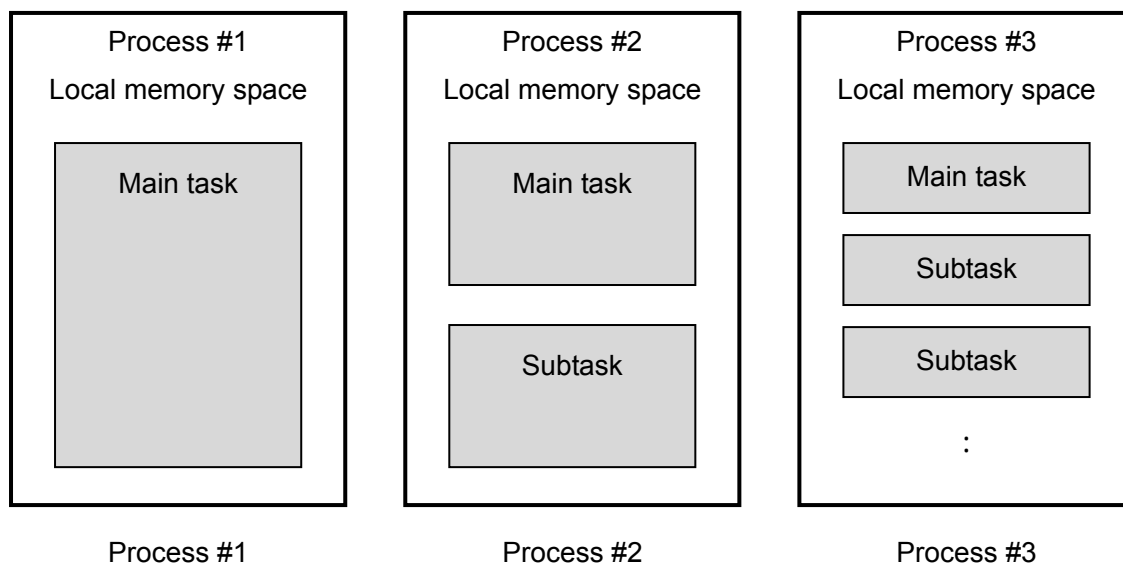
A process is a unit used by AMP TKSE to manage programs. Multiple processes can simultaneously exist on a single system. Each process has an independent local memory space and an execution environment, and runs in parallel with other processes.

A process is created when an execution program file on a file system is read using a system call for process creation. Each process that has been created is given a unique process ID by which it is identified. A process ID is a positive integer.

A process has one or more tasks. A task is an execution unit of a program. Tasks are scheduled for execution according to their task priorities.

A task that goes to READY state immediately after process is created is called a main task. There can be only one main task in each process and, when the main task exits, the entire process immediately exits. In addition to the main task, subtasks can be created by invoking the system call for task creation. One or more subtasks can be created in one process. Even when a subtask exits, the process does not exit. A main task and subtasks are collectively called a task. A unique task ID is given to a task at the time of creation, and the individual task is identified by this ID. A task ID is a positive integer.

Tasks in the same process share the local memory space.



[Figure 2] Relationship between processes and main task/subtasks

A process that has created another process is called a parent process and the created process is called a child process. All the processes have a parent process. However, the initial process, the one created first at system startup, does not have a parent process. The entire system, therefore, consists of processes formed in a tree structure with the initial process defined as the root.

When Process A in a tree structure exits, the child processes of Process A get a new parent process, which is the parent process of Process A. The general tree structure will be thus maintained. One exception is that the initial process exits and its child processes no longer have a parent process.

2.2.2 Address Space of Process

A space that a program can access using a specified address is called an address space. A 32-bit address space has addresses from 0x00000000 to 0xFFFFFFFF, which can be used to access memory or I/O device mapped to each of these addresses.

Address spaces consist of physical and logical address spaces. A physical address space is defined at the time of system hardware design. A logical address space is virtually managed using functions such as MMUs. Processes of AMP TKSE normally use only the logical address space.

Part of the address space mapped to memory is called a memory space. However, actual physical memory may not be allocated to all the addresses of a memory space because virtual memory is supported by AMP TKSE.

To enable access to a memory space by allocating actual memory, the system call (`tkse_get_mbk`) or the library (`malloc API`) of the AMP TKSE memory management functions shall be used. A block of memory with contiguous logical addresses thus assigned is called a "memory area." Processes allocate and release the memory area as required to enable access to memory.

AMP TKSE manages virtual memory using a page file on the file system. This enables use of a larger memory area than the actual physical memory size. Since page-in and page-out from the memory area is automatically executed by the memory management of AMP TKSE, an application can use the memory area without being aware of whether the memory area exists in the physical memory or not. If realtime memory access is required, page-out of the target area can be prohibited by specifying it (to be) memory-resident. The specified resident memory area always exists in the physical memory.

AMP TKSE has the following four types of memory spaces:

- Local memory space
- Shared memory space
- System memory space
- Shared memory space between kernels

Local memory space refers to independent memory space for each process. The code area and data area to be used by the process is normally placed in the local memory space.

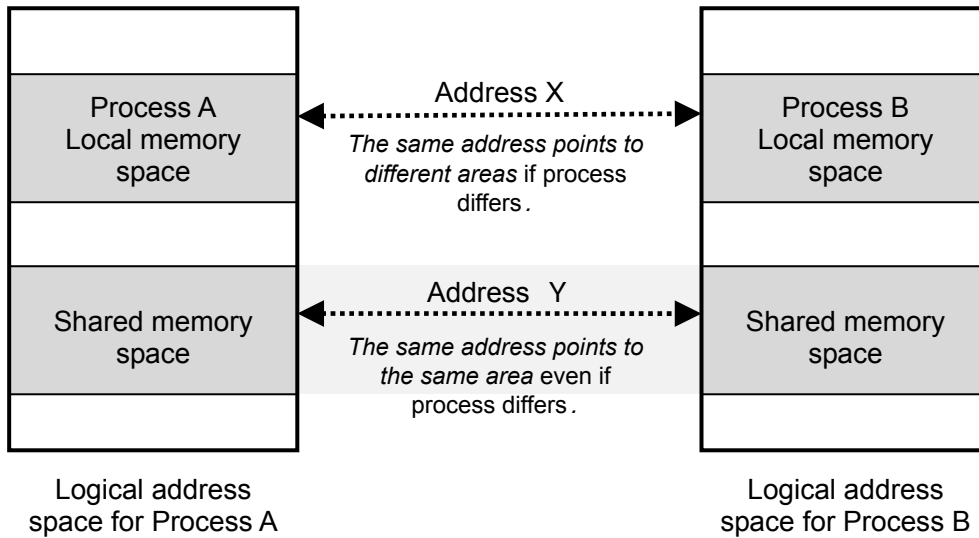
The local memory space of one process cannot be accessed by another process. When a process tries to access the local memory space using an address belonging to the local memory space of another process, access to its own process area occurs if the memory area for this process is allocated at this address in its own local memory space. A memory protection exception occurs otherwise.

Addresses of an area in the local memory space are unique to each process. Processes may allocate areas with overlapping addresses, but they actually point to different areas. If Process A allocates an area in the local memory space, Address X in this area cannot be used by Process B. Address X for Process A and Address X for Process B may have the same value but actually point to different areas.

Shared memory space is accessible from any process of its own AMP TKSE. This space can be used to pass data between processes.

Addresses of an area allocated in shared memory space are common to all the processes. If Process A allocates

an area in the shared memory space which has Address Y, Process B can read the same area by accessing Address Y.



[Figure 3] Local memory space and shared memory space

System memory space refers to special memory space that AMP TKSE uses internally. This space, intended for use by a system program or driver, must not be used by a general application. If a process accesses an area of the system memory space, a memory protection exception occurs in the same way as when it accesses the local memory of another process.

The shared memory space between kernels is accessible from any process of all AMP TKSE. This space can be used for sending and receiving data between kernels.

The address of the area allocated in the shared memory space between kernels is common among processes. For address Y of the space allocated by process A in the shared memory space between kernels can be accessed by accessing address Y from process B as well.

2.2.3 Process State and Task State

A task has a task state according to its state of operation. A process state refers to the task state of the main task of each process.

A task state is any of the following five basic states. These task states conform to the task state of AMP T-Kernel. However, a task cannot be put in SUSPEND state. Additionally, only a subtask can be put in DORMANT state. No main task can be put in DORMANT state. No running task can be put in DORMANT state.

(1) RUN state

This means that the task is currently being executed. At any one time, there can be one task in RUN state at most.

(2) READY state

This means that the task, which is ready for execution, cannot be executed because another task with a higher precedence is being executed.

When the task in RUN state goes to READY or WAIT state, a new task goes to RUN state from among the tasks in READY state in accordance with the order of precedence.

Dispatch refers to an operation in which a CPU resource is allocated to a task in READY state and the task goes to RUN state. Preempt refers to an operation in which a task in RUN state releases the CPU resources and goes to READY state.

(3) WAIT state

This means that the execution of the task is temporarily suspended because a system call is invoked to suspend the execution of the task itself.

(4) DORMANT state

This means that the task has not yet been started or has completed execution.

While a task is in DORMANT state, information regarding its execution state is not saved. When a task in DORMANT state is started and goes to READY state, execution of the task starts from the task start address. Only a subtask can go to DORMANT state.

(5) NON-EXISTENT state

This means that the task has not yet been created or has been deleted.

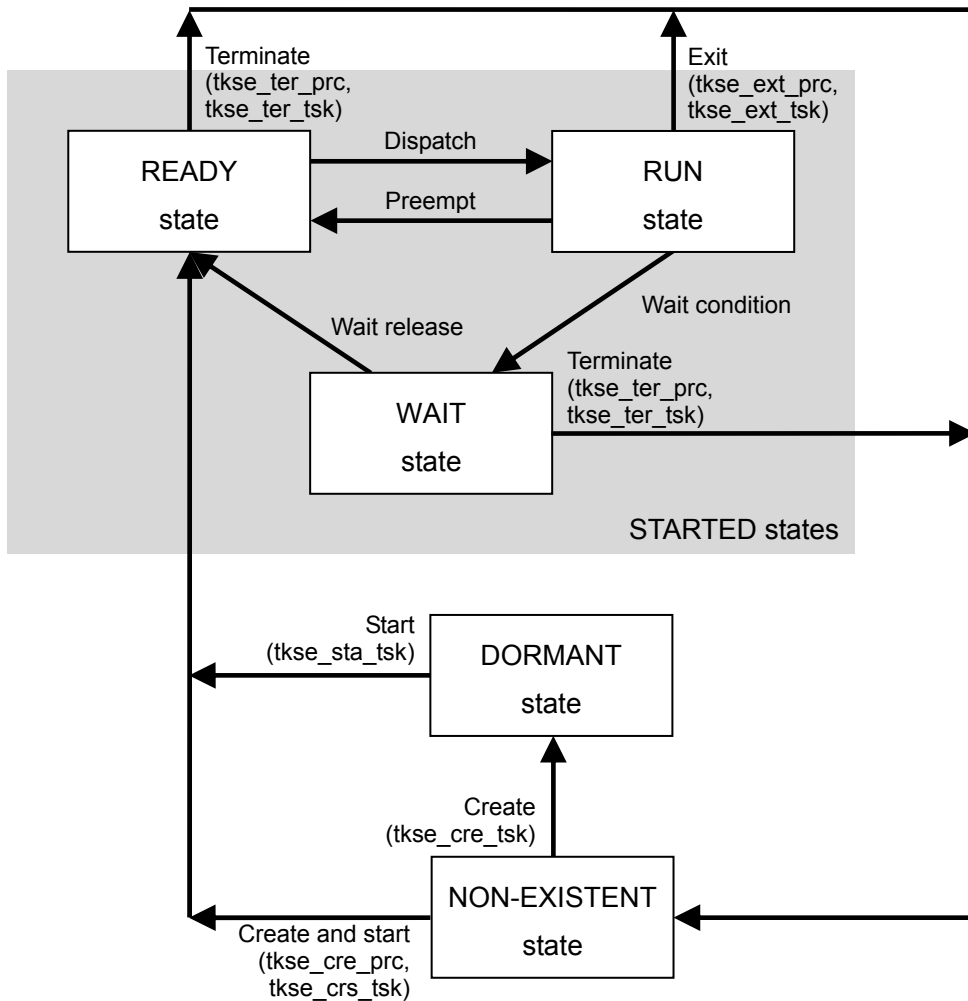
The NON-EXISTENT state is a virtual state. A task in NON-EXISTENT state is actually not registered in the system.

The following shows the task state transition for a general implementation. Depending on the implementation, there can be state transitions not shown in this figure or transient states that do not fall into any of the categories provided.

When a task going to READY state has higher precedence than the currently running task, a dispatch may occur at the same time as the task goes to READY state, and it may make an immediate transition to RUN state. In such a case, the task that has been in RUN state up to this point is said to have been preempted by the new task going to RUN state. Also note that, even if the explanation of a system call function describes that a task "goes to READY state," it may immediately go to RUN state depending on the task precedence.

"Task start" refers to the transition of a task in DORMANT state into READY state. Therefore, all other states than DORMANT and NON-EXISTENT states may be called "STARTED" state collectively. Task exit refers to the transition of a task in STARTED state into DORMANT state.

"Task wait release" refers to the transition of a task in WAIT state into READY state. A factor that releases WAIT state is called a task wait release factor.



[Figure 4] Task State Transitions

2.2.4 Process/Task Priority and Scheduling

Each task has its own task priority. The task priority of the main task of a process is called a process priority. Each subtask also has priority; Subtask priority can be set to a different value than process priority. Priority should be set at the time of creating a process or subtask. Priority can be changed dynamically while a task is running.

Priority can be set to a range of values from 0 to 255 (with 0 being the highest priority). Tasks are classified into three priority groups according to their priority values, each of which is given a different scheduling. AMP TKSE basically offers two types of scheduling: Absolute priority scheduling and round robin scheduling.

With absolute priority scheduling, the higher the task priority is, the higher the task precedence is. Therefore, while a task is in RUN state, another task with a lower priority than the former never goes to RUN state. This scheduling is basically the same as the scheduling of AMP T-Kernel.

With round robin scheduling, tasks go to RUN state in turn without regard to the task priorities. A task priority is effective in this scheduling frequency. More specifically, the higher the priority is, the more run time is allocated to a task (time during which a task can stay in RUN state). After the allocated run time elapses, the task precedence becomes the lowest, and another task goes to RUN state. In other words, a task with the lowest priority is executed without fail eventually.

Depending on the priority values, tasks are classified into the following three priority groups:

A. Absolute priority group (Priority: 0 to 127)

Tasks in this group are subject to absolute priority scheduling based on task priorities (with 0 being the highest priority).

The task with the highest priority goes to RUN state. Tasks with lower priority do not run.

However, for tasks with the same priority, scheduling is performed equally among tasks with the same priority at regular intervals by the round robin method.

B. Round robin group 1 (Priority: 128 to 191)

Tasks in this group are scheduled in a round robin fashion (with 128 being the highest priority).

This group has a lower precedence than the absolute priority group. If any task in the absolute priority group is in RUN or READY state, no task in this group is executed. If no task in the absolute priority group is in RUN or READY state, the eventual execution of a task with a low priority in this group is generated.

C. Round robin group 2 (Priority: 192 to 255)

Tasks in this group are scheduled in a round robin fashion (with 192 being the highest priority).

This group has a lower precedence than other groups (the absolute priority group and round robin group 1). If any task in other groups is in RUN or READY state, no task in this group is executed. If no task in other groups is in RUN or READY state, the eventual execution of a task with a low priority in this group is ensured.

Actual scheduling is executed as follows:

1. If there are tasks in READY state in the absolute priority group, the one with the highest priority goes to RUN state and is executed. If not, go to step 2.
If there are multiple tasks with the highest priority, they are scheduled equally and periodically in a round robin fashion.
2. If there are tasks in READY state in round robin group 1, a task is selected according to the relative precedences of the tasks, and then goes to RUN state and is executed. This task may not necessarily be the highest priority). If not, go to step 3.
3. If there are tasks in READY state in round robin group 2, a task is selected according to the relative precedences of the tasks, and then goes to RUN state and is executed. (This task may not necessarily be the highest priority).

2.2.5 Execution Environment of Process

A process retains the following information as an execution environment:

- Process IDs of this process, parent process, and child process

- Process/Task priorities
- Current work files (Standard file management)
- Open files (Standard file management, standard I/O)
- Message queue (Interprocess messages)

The execution environment immediately after a process is created is set up as follows:

- Process ID of this process: ID assigned at the time of creation
- Process ID of parent process: ID of process that created this process
- Process ID of child process: None
- Process/Task priority: Priority specified at the time of creation
- Current work files: Work file of parent process at the time of creation
- Open files: None
- Message queue: Empty

Kernel objects such as semaphores can be associated with a process that created such objects by specifying an attribute (`TA_DELEXIT`, `TA_PLOCAL`) at the time of creation. An object associated with a process is automatically deleted when the process exits.

2.2.6 User Process and System Process

There are two types of processes: User process and system process. A process can be specified as a user or system process by specifying an attribute when the process is created.

A user process can use all the functions of AMP TKSE. A system process can use the functions available to a user process and directly use system calls (`tk_xxx_yyy`, etc.) of AMP T-Kernel.

A system process is intended for a use close to the system core, e.g., in combination with a debugger or upper system. In principle, a general application shall be specified as a user process.

2.2.7 Creating a Process

A process can be created by invoking a system call, `tkse_cre_prc`, specifying an execution program file of a process and process creation message.

A process creation message is a message passed from a parent process to its child process at process creation.

A process creation message has a message structure identical to that of an interprocess message.

```

typedef struct {
    W msg_type;          /* Message type */
    W msg_size;         /* Message size (in bytes) */
    UB msg_body[n];     /* Message body (msg_size bytes) */
} MESSAGE;

```

When a process has been successfully created, the main task of the process is started. At this time, the main task function receives a process creation message as an argument.

Either of two methods of receiving a message can be selected: Receiving message data directly or receiving individual components into which message data is split by assuming that it consists of character strings delimited with blanks. According to the use by the user, the name definition for the main task function should be selected from those shown in the following. However, only one of the names can be defined at the same time.

(1) Format 1

```

W MAIN (MESSAGE *msg)
/* MESSAGE *msg; Pointer to a message */
{
    /* Program execution code */
    return exit-code;
}

```

When the name of a main task function is defined as `MAIN`, a process creation message `msg` is directly received as a function argument. At this case, there is no limit on the value of message type `msg_type`.

(2) Format 2

```

W main (W ac, TC **argv)
/* W ac; Number of string items */
/* TC **argv; Pointer to array of points to string items */
{
    /* Program execution code */
    return exit-code;
}

```

When the name of a main task function is defined as `main`, message data `msg_body` of a process creation message is regarded as one TRON character code string that is delimited with space character `TK_KSP` and ends with `TNULL`. In this case, the number of items delimited with space characters is passed to the `main` task function argument `ac` and pointers to strings in each item are passed to argument `argv` as a pointer array.

If `msg_body` does not end with `TNULL`, the termination character of `msg_body` is replaced with `TNULL` before argument analysis processing is executed. At this case, the termination character of `msg_body` is lost.

When Format 2 is used, message type `msg_type = 0` must be specified. If `msg_type ≠ 0` is specified, `ac = 0` and `*argv = NULL` are always set regardless of the content of `msg_body`, and therefore no message can be received.

The process exits when processing returns from main task function `MAIN` or `main`. This is equivalent to exiting from a process by `tkse_ext_prc`.

2.2.8 Combination with AMP T-Kernel Programs

Applications that run in AMP TKSE can run in combination with AMP T-Kernel programs. Applications can access mainly the following two types of AMP T-Kernel programs:

- Device drivers
Device drivers control various devices connected to the system.
They are accessed using the device management function of AMP TKSE.
- Subsystems
Subsystems are used by various middleware to add functions to the system.
They are accessed using extended SVC provided by subsystems.

These AMP T-Kernel programs are collectively called system programs. System programs run in the same system memory space as AMP T-Kernel.

System programs can be placed in the system memory space by linking them directly to AMP T-Kernel. They can also be dynamically loaded and unloaded by applications.

An application can load a system program by issuing `tk_lod_spg` with the system program executable file being one of its arguments. Since the area into which it is loaded is dynamically allocated, the system program must be created in a relocatable format. If the location address of a system program stored in an executable file is a logical address out of the range managed by the operating system, it is loaded at a fixed address according to the location information of the executable file.

After the system program is loaded, execution starts with the `main` function written in the following pattern. Unlike process creation, the `MAIN` function cannot be used.

```

W main (W ac, TC **argv)
/* W ac; Number of string items */
/* TC **argv; Pointer to pointer array of string items */
{
    if (ac >= 0) {
        /* Program load processing */
    } else {
        /* Program unload processing */
    }
    return exit-code;
}

```

Argument `arg` used when `tkse_lod_spg` loads a system program is regarded as one TRON code string delimited with spaces and terminated with `TNULL`. The number of items delimited with space characters is passed to the `main` function argument `ac` and pointers to strings in each item are passed to argument `argv` as a pointer array. At the time of loading, `ac >= 0` is always set.

`tkse_unl_spg` unloads a system program that has been loaded. In the same way as for loading, the `main` function is invoked. However, `ac < 0` is set at the time of unloading and therefore each of loading or unloading processing is terminated selectably after evaluating the value of `ac`.

The `main` function is executed as the quasi-task portion of a task that invoked `tkse_lod_spg`. Since the `main` function can use T-Kernel API, the definition or deletion of a subsystem in the case of a subsystem or the registration or deletion of a device in the case of a device driver is performed here.

The `main` function must not change the status of any of its tasks such as exiting because it may affect the RUN state of the invoking task.

2.3 Synchronization and Communication

2.3.1 Interprocess Synchronization and Communication

AMP TKSE provides the following functions to execute interprocess communication:

(1) Interprocess message

The interprocess message function sends a data structure called a message from a sending process to a receiving process to realize one-to-one interprocess communication. This function can also be used for interprocess synchronization.

A message sent by Send Message (`tkse_snd_msg`) is stored in a message queue of a receiving process. A message queue is unique to each process, and automatically created and initialized when a process is created. The receiving process executes Receive Message (`tkse_rcv_msg`) to retrieve a message stored in the message queue of this process. Asynchronous message reception can also be executed if a message handler is defined for the receiving process. In this case, when the receiving process receives a message, the message handler is started while interrupting the main task.

Interprocess message is used not only as a means of interprocess communication but also of delivering information from the system to a process. For example, when a child process exits, a child process exit message is sent from the system to its parent process. Such a message sent by the system is called a system message.

Messages between processes can be sent to the processes of AMP TKSE on other T-Kernels.

(2) Global name

The global name function allows multiple processes to share four-byte data to which an arbitrary name called a global name has been assigned.

Since each process has an independent local memory space, multiple processes cannot share data with each other using, for example, global variables in a local memory space. Shared memory and message buffer can be used to share data. To use these functions, however, it is necessary to first share the addresses of an area of shared memory or the IDs of OS objects such as message buffer. The global name function is intended to share such addresses and IDs.

Although the use of the global name function is intended for a sharing of addresses and IDs, any four-byte data can be shared.

The global name can be used only on AMP TKSE of a processor.

(3) Shared memory

Interprocess communication using shared memory is a method for passing data using the shared memory space described above.

This method is used to allow multiple processes to access large quantities of data. Instant passing of data is possible because no data copy is executed. However, considerations must be given to the absence of access protection and the necessity of combined use of other functions for synchronization and exclusive control.

Shared memory can be used only on AMP TKSE of a processor.

Interprocess synchronization can be executed using the interprocess message function. If more detailed synchronization at the task level is required, the intertask synchronization and communication function can be used.

2.3.2 Intertask Synchronization and Communication

In order to conduct synchronization and communication between tasks, AMP TKSE provides a task synchronization and communication function and a task-dependent synchronization function.

(1) Intertask synchronization and communication functions

Intertask synchronization and communication are achieved using OS objects provided for synchronization and communication. The functions provided by these OS objects are called the intertask synchronization and communication functions. (For more details, refer to the next section.)

The task communication function can be used for AMP TKSE objects on other T-Kernels.

(2) Task-dependent synchronization functions

Synchronization among tasks can also be achieved by directly manipulating the states of other tasks instead of using the intertask synchronization and communication functions. The functions used to achieve synchronization through control of the states of other tasks are called the task-dependent synchronization functions.

The task-dependent synchronization functions available in AMP TKSE are the Wakeup task, Sleep task and cancellation of them.

Task-dependent synchronization functions can be used only for tasks within the same process. The task state of tasks in other processes cannot be operated.

2.3.3 Intertask Synchronization and Communication Function

The following objects can be used as task synchronization and communication functions.

- Semaphores
- Mutexes
- Event flags
- Mailboxes
- Message buffers
- Rendezvous ports

In order to use these task synchronization and communication functions, the target object of the function is first created. A specific object ID is allocated to the created object. By specifying this object ID, synchronization and communication between tasks are conducted. The Object ID is unique in the entire AMP system. In other words, it does not overlap with the IDs of objects of other AMP TKSE.

Each object specifies any of the following access attributes during creation.

(1) Global Attribute

Objects with the global attribute are accessible from all tasks including the tasks of other AMP TKSE.

(2) Kernel Local Attribute

Objects with the kernel local attribute can be accessed from all tasks of AMP TKSE to which the object belongs. They cannot be accessed from tasks of other AMP TKSE.

(3) Process Local Attribute

Objects with the process local attribute can be accessed only from within the process to which the task which created the object belongs. They cannot be accessed from tasks of other processes.

By using objects with the global attribute, synchronization and communication between tasks can be performed between applications of AMP TKSE. In other words, synchronization and communication becomes possible between kernels.

The mailbox cannot be used between processes, and only the process local attribute can be specified.

The specifications for the task synchronization and communication function of AMP TKSE basically conform to the specifications for the task synchronization and communication function of AMP T-Kernel. However, since object IDs are independently managed by AMP TKSE, the object IDs of objects created with AMP TKSE cannot be used with AMP T-Kernel as they are. Moreover, object IDs for objects created with AMP T-Kernel cannot be used with AMP TKSE either. In addition, there are some restrictions on the attribute specification when objects are created (For details of the specification, refer to the explanation of each system call).

While the main task is in WAIT state because of the use of the task synchronization and communication function, if the message handler of the process interrupts, the WAIT state of the task is released and the system call returns error code E_DISWAI.

2.4 Object Management

2.4.1 Retrieving Object ID Number

Processes as well as synchronization and communication objects are identified by the ID number. Since an ID number is automatically allocated when the object is created, the means to know the ID number of a target object from the application is necessary. In AMP TKSE, the ID number can be retrieved from the name given to an object.

Synchronization and communication objects can be given object names when they are created. The object name must be unique for the same type of object. However, for objects with the kernel local attribute, the name must be unique only within its own kernel. Similarly, the name of objects with the process local attribute must be unique only within its own process.

The ID number of an object can be retrieved with the object name by the object management function. Object ID

number retrieval can be performed for objects of other AMP TKSE as well. However, only objects, which can be accessed from the process and the task, can be retrieved. In other words, only the following objects are subject to retrieval.

- Objects with a global attribute
- Objects with a kernel local attribute within the same kernel
- Objects with a process local attribute within the same process

The process ID number can be retrieved by the object name given during creation as in the case of the synchronization and communication object. The process is treated as if it was given a global attribute although the access attribute cannot be specified during creation.

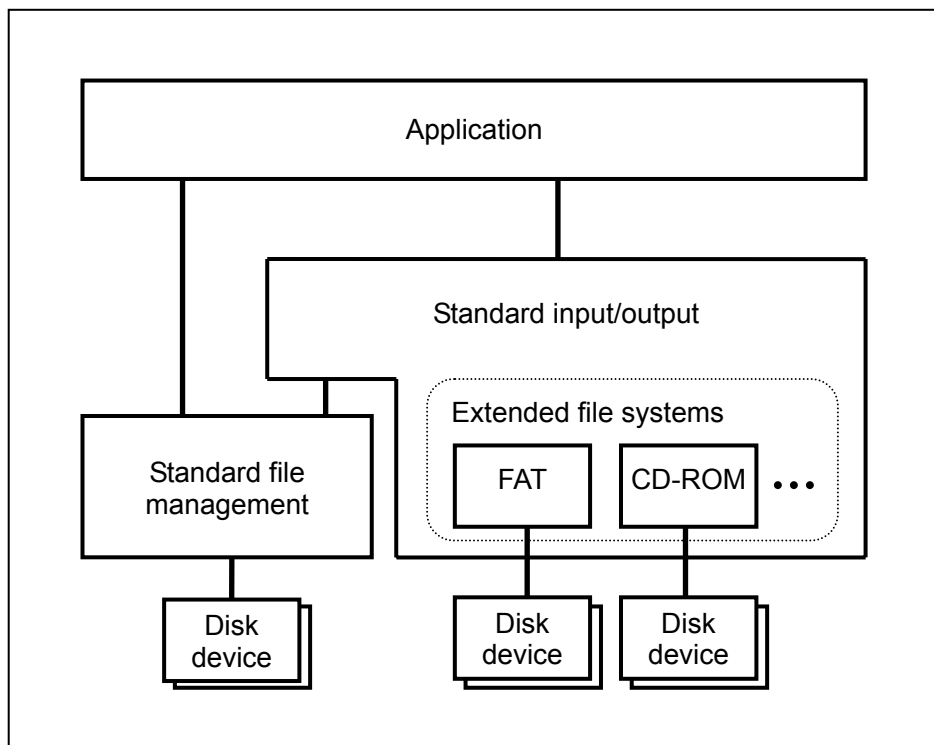
2.5 Standard File Management and Standard Input/Output Functions

2.5.1 File Management of AMP TKSE

AMP TKSE has the file management functions that permit the use of a disk device registered in AMP T-Kernel as file system.

The file management functions consist of the standard file management function and the standard input/output function: The former is used to directly manipulate the T-Kernel standard file system (hereinafter called standard file system), and the latter is used to handle various file systems including the standard file system in a uniform manner.

The standard input/output function can handle not only the standard file system but also file systems in other formats. These file systems in other formats are called extended file systems. The file formats supported in the specifications of the current version are the FAT12, FAT16, and FAT32 file systems and the CD-ROM (ISO9660 Level1) file system. A different file system can also be embedded in the standard input/output as an extended file system.



[Figure 5] Overview of File Management

To use a disk device as a file system, it is necessary to connect the file system first. The connected file system has a unique connection name, which is then used to manipulate files on the file system. The standard file management and the standard input/output permit simultaneous connection of multiple different file systems. A file system must be connected before starting a process or system program from an executable file or performing virtual memory management using a page file.

2.5.2 Standard File Management Function

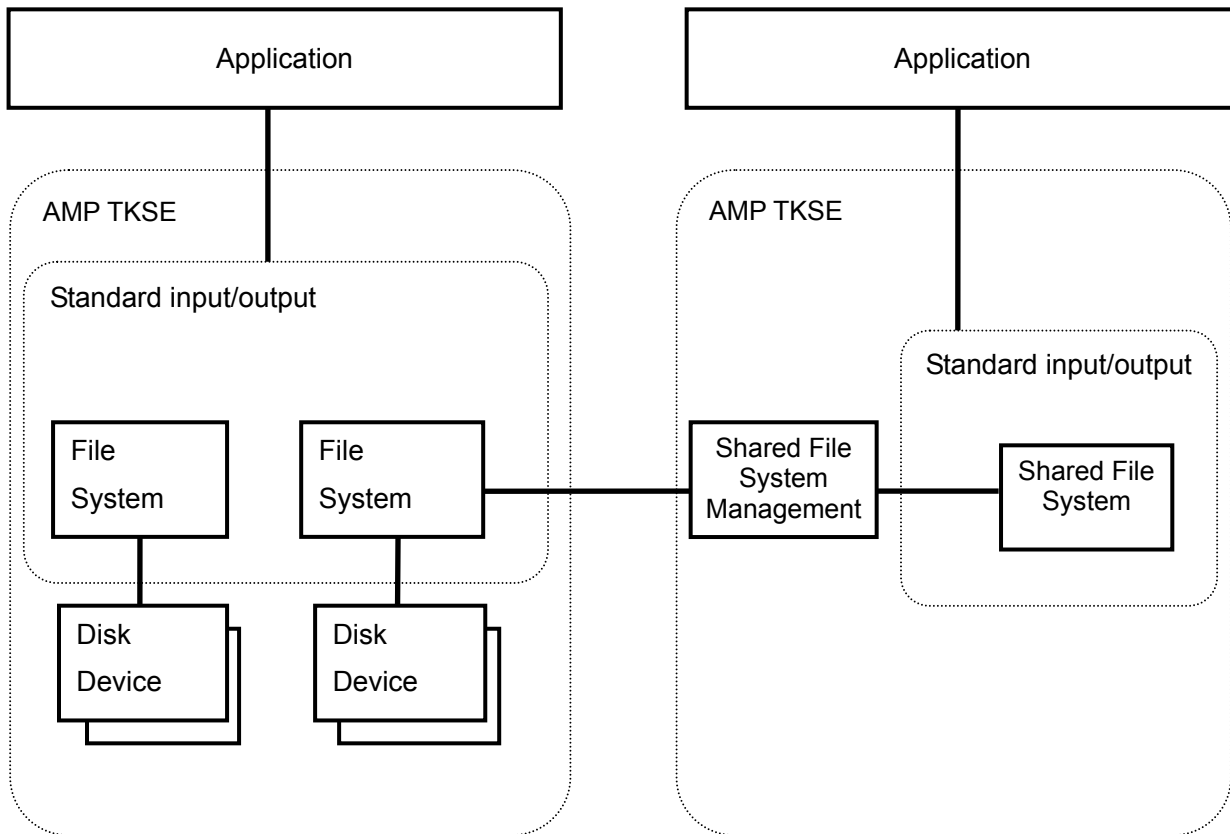
The standard file management function handles the standard file system with a hypertext-based network structure. The standard file management function is used for handling of real and virtual objects and file records, which is the function of the standard file system.

2.5.3 Standard Input/Output Function

The standard input/output function realizes file access from applications using common system calls irrespective of differences between specifications of file systems. However, restrictions on file name lengths, maximum file sizes, and others of the original file systems also apply to the standard input/output.

2.5.4 Shared Filesystem Management

AMP TKSE has a shared filesystem management function for sharing filesystems with other AMP TKSE. Due to this shared function, applications can use the standard input/output function without distinguishing the filesystem of its own AMP TKSE from the file systems of other AMP TKSE. In other words, the applications can handle the disk devices of other processors as a filesystem in the same way as the disk devices of its own processor.



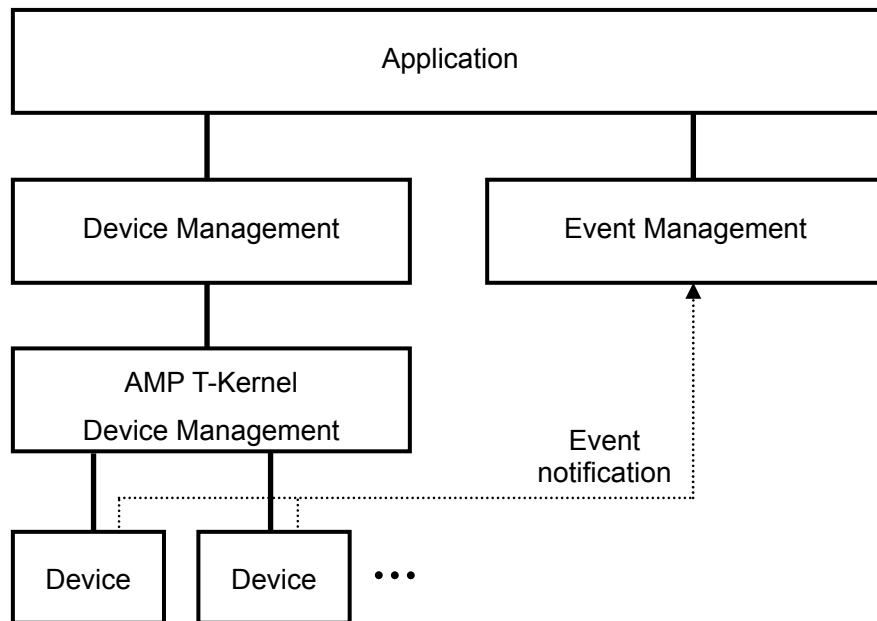
[Figure 6] Shared File Management

As for sharing filesystems, since a large amount of data is received and sent between kernels, the existence of shared memory between kernels is a prerequisite.

2.6 Device Management and Event Management Functions

2.6.1 Access to T-Kernel Device and Event Notification

AMP TKSE provides the device management function that permits access to devices registered in AMP T-Kernel, and the event management function that allows applications to receive event notifications sent from devices.



[Figure 7] Overview of Device Management and Event Management

2.6.2 Device Management Function

Device management function permits access to the AMP T-Kernel device management function from AMP TKSE. The actual manipulation and management of devices are executed by AMP T-Kernel. Devices can be registered only from AMP T-Kernel. Devices cannot be registered or unregistered from AMP TKSE. Device management only targets the devices registered in its own AMP T-Kernel. It cannot manipulate the devices of other AMP T-Kernel.

2.6.3 Event Management Function

Event management function allows applications to receive event notifications generated asynchronously by devices. Event notifications from devices are converted into a data structure called an event, and are stored sequentially in the event queue of event management. One event queue exists for each AMP TKSE and only events from devices registered in its own AMP T-Kernel are stored. Events from devices registered in other AMP

T-Kernel cannot be handled.

Since only one event queue exists per AMP TKSE, the event management function cannot be used from multiple processes at the same time.

An application can retrieve an event stored in the event queue. It can also receive an event as a message.

The main purpose of event management is to realize interactive human interfaces. Therefore, event management is designed on the assumption that it is used to send event notifications from devices such as keyboards, and pointing devices to applications as *events*. However, device events, extended device events, application events, and other events can also be used to send event notifications from other devices to applications.

3. AMP T-Kernel Standard Extension Common Specifications

3.1 Data Types

3.1.1 Basic Data Types

```

typedef      char          B;          /* Signed 8-bit integer */
typedef      short        H;          /* Signed 16-bit integer */
typedef      int           W;          /* Signed 32-bit integer */
typedef      unsigned char UB;        /* Unsigned 8-bit integer */
typedef      unsigned short UH;       /* Unsigned 16-bit integer */
typedef      unsigned int  UW;        /* Unsigned 32-bit integer */

typedef      char          VB;         /* 8-bit data without a fixed type */
typedef      short        VH;         /* 16-bit data without a fixed type */
typedef      int           VW;         /* 32-bit data without a fixed type */
typedef      void          *VP;        /* pointer to data without a fixed type */

typedef      volatile B    _B;        /* volatile declaration */
typedef      volatile H    _H;
typedef      volatile W    _W;
typedef      volatile UB   _UB;
typedef      volatile UH   _UH;
typedef      volatile UW   _UW;

typedef      int           INT;        /* Signed integer of processor bit width */
typedef      unsigned int  UINT;      /* Unsigned integer of processor bit width */

typedef      INT           ID;         /* ID in general */
typedef      INT           MSEC;      /* time (milliseconds) in general */
typedef      void          (*FP)();   /* function address in general */
typedef      INT           (*FUNCP)(); /* function address in general */

#define      LOCAL         static     /* Local symbol definition */
#define      EXPORT
#define      IMPORT         extern   /* Global symbol reference */

```

/*

* Boolean values

* TRUE = 1 is defined below, but any value other than 0 is TRUE.

* A decision such as `bool == TRUE` must be avoided for this reason.

* Instead, use `bool != FALSE`.

*/

```
typedef      INT          BOOL;
#define      TRUE         1          /* True */
#define      FALSE        0          /* False */
/*
 * TRON code
 */
typedef      UH           TC;         /* TRON character code */
#define      TNULL        ((TC)0)    /* TRON code string terminating character */
```

- * The difference between VB and B, between VH and H, and between VW and W is that the former means only the bit width is known, not the contents of the data type, whereas the latter clearly indicate integer type.
- * Processor bit width must be 32 bits or more. INT and UINT must therefore always have a width of 32 bits or more.
- * BOOL defines TRUE = 1, but any value other than 0 is also TRUE. For this reason, a decision such as `bool == TRUE` must be avoided. Instead, use `bool != FALSE`.

[Additional Notes]

Parameters that do not take negative values in principle are also signed integer (INT) data types. This is in keeping with the overall TRON rule that integers should be treated as signed numbers to the extent possible. As for the timeout (TMO tmout) parameter, its being a signed integer permits the use of TMO_FEVR (= -1) having special meaning. Parameters with unsigned data type are those treated as bit patterns (object attribute, event flag, etc.).

3.1.2 Other Defined Data Types

The following names are used for other data types that appear frequently or have special meaning, in order to make the parameter meaning clear.

```
typedef      INT          FN;         /* Function code */
typedef      INT          RNO;        /* Rendezvous port number */
typedef      UINT         ATR;        /* Object/Handler attributes */
typedef      INT          ER;         /* Error code */
typedef      INT          PRI;        /* Priority */
typedef      INT          TMO;        /* Timeout */
typedef      UINT         RELTIM;     /* Relative time */
typedef      struct systim {          /* System time */
                W          hi;        /* High 32 bits */
                UW         lo;        /* Low 32 bits */
} SYSTIM;
```

```

/*
 * Common constants
 */
#define NULL 0 /* Null pointer */
#define TA_NULL 0 /* No special attributes indicated */
#define TMO_POL 0 /* Polling */
#define TMO_FEVR (-1) /* Eternal wait */

```

* A data that can take the value of two or more data types is declared using its main data type. For example, the value returned by `tkse_cre_prc` can be a process ID or error code. However, since it is mainly a process ID, the data type is ID.

3.2 Error Codes

3.2.1 Overview

System call return codes are in principle to be signed integers. When an error occurs, a negative error code is returned; and if processing is completed normally, `E_OK` (= 0) or a positive value is returned. The meaning of the returned values for normal completion is specified separately for each system call. An exception to this principle is that there are some system calls that do not return a return code when called.

A system call that does not return a return code is declared in the C language API as having no return code (that is, a void type function).

An error code consists of the main error code and sub error code. The low 16 bits of the error code are the sub error code, and the remaining high bits are the main error code. Main error codes are classified into error classes based on the necessity of their detection, and the circumstances in which they occur and other factors.

```

#define MERCD(er) ((ER)(er) >> 16) /* Main error code */
#define SERCD(er) ((H)(er)) /* Sub error code */
#define ERCD(mer, ser) ((ER)(mer) << 16 | (ER)(UH)(ser))

```

3.2.2 List of Error Codes

The following shows error codes of AMP TKSE. Main error codes from 0 to -255 are error codes compatible with T-Kernel and have the same meaning as those for AMP T-Kernel. Error codes from -256 and downward are error codes unique to Standard Extension. There is no error code unique to AMP TKSE.

Error codes in a range not defined as an error class are reserved for the purpose of future expansions.

Normal Completion Error Class (0)

E_OK 0 Normal completion

Internal Error Class (-5 to -8)

E_SYS ERCD(-5, 0) System error

An error of unknown cause affecting the system as a whole.

E_NOCOP ERCD(-6, 0) The specified co-processor cannot be used

This error code is returned when the specified co-processor is not installed in the currently running hardware, or abnormal co-processor operation was detected.

Unsupported Error Class (-9 to -16)

E_NOSPT ERCD(-9, 0) Unsupported function

When some system call functions are not supported and such a function was called, error code E_RSATR or E_NOSPT is returned. If E_RSATR does not apply, error code E_NOSPT is returned.

E_RSFN ERCD(-10, 0) Reserved function code number

This error code is returned when it is attempted to execute a system call specifying a reserved function code (undefined function code), and also when an attempt to execute an undefined extended SVC handler (when the function code is positive) is made.

E_RSATR ERCD(-11, 0) Reserved attribute

This error code is returned when an undefined or unsupported object attribute is specified. Checking for this error may be omitted if system-dependent optimization is implemented.

Parameter Error Class (-17 to -24)

E_PAR ERCD(-17, 0) Parameter error

Checking for this error may be omitted if system-dependent optimization is implemented.

E_ID ERCD (-18, 0) Invalid ID number

E_ID is an error code that occurs only for objects having an ID number.

Error code E_PAR is returned when a static error is detected because, for example, the specified ID number is a reserved number or out of range of interrupt definition numbers.

Call Context Error Class (-25 to -32)

E_CTX ERCD(-25, 0) Context error

This error code indicates that the specified system call cannot be issued in the current context (the context must be the task portion/task-independent portion or handler RUN state).

This error is always returned whenever a system call is issued in a semantically incorrect context, for example, when a system call that sends its own task into WAIT state is issued from a task-independent portion. This error is returned also for other system calls when, due to implementation limitations, they cannot be issued in a given context (such as an interrupt handler).

E_MACV ERCD(-26, 0) Memory cannot be accessed; memory access privilege error

Error detection is implementation-dependent.

E_OACV ERCD(-27, 0) Object access privilege error

This error code is returned when a user task tries to manipulate a system object.

The definition of system objects and error detection are implementation-dependent.

E_ILUSE ERCD(-28, 0) System call illegal use

Resource Constraint Error Class (-33 to -40)

E_NOMEM ERCD(-33, 0) Insufficient memory

This error code is returned when there is insufficient memory (no memory) for allocating an object control block space, user stack space, memory pool space, message buffer space or the like.

E_LIMIT ERCD(-34, 0) System limit exceeded

This error code is returned when an attempt to create more objects than the system allows is made.

Object State Error Class (-41 to -48)

E_OBJ ERCD(-41, 0) Invalid object state

E_NOEXS ERCD(-42, 0) Object does not exist

E_QOVR ERCD(-43, 0) Queuing or nesting overflow

Wait Error Class (-49 to -56)

E_RLWAI ERCD(-49, 0) WAIT state released

E_TMOUT ERCD(-50, 0) Polling failed or timeout

E_DLT ERCD(-51, 0) The object being waited for was deleted

E_DISWAI ERCD(-52, 0) Wait released by wait disabled state

Device Error Class (-57 to -64) (T-Kernel/SM)

E_IO ERCD(-57, 0) IO error

* Error information specific to individual devices may be defined in E_IO sub-codes.

E_NOMDA ERCD(-58, 0) No media

Status Error Class (-65 to -67) (T-Kernel/SM)

E_BUSY ERCD(-65, 0) Busy

E_ABORT ERCD(-66, 0) Processing was aborted

E_RDONLY ERCD(-67, 0) Write protected

Domain Error Class (-68 ~ -70) (MP T-Kernel)

E_DOMAIN ERCD(-68, 0) Domain error

This error code indicates that an operation is not permitted due to its inter-domain nature, i.e., when the operation was attempted for an object that belongs to another domain.

E_ONAME ERCD(-69, 0) Object name error

This error code indicates that the specified object name has already been used in the domain.

E_DACV ERCD(-70, 0) Access protection error

This error code indicates that the operation is not permitted due to access protection of an object.

Error Class Between Processors (-71 ~ -73) (MP T-Kernel)

E_IPC ERCD(-71, 0) Interprocessor communication error

This error code indicates that a failure occurred in some sort of communication between processors during the execution of a system call, and the execution result of the system call was unknown.

This error occurs when reply from another processor could not be received normally. When this error code is returned, the result of the system call is not guaranteed. There is also the possibility that the requested operation is executed on another processor. If the cause of failure of the requested operation can be elaborated more, an E_IPCA or E_IPCS which is described below is returned instead of this error code.

E_IPCA ERCD(-72, 0) Absolute interprocessor communication error

This error code indicates that a failure occurred in some sort of communication between processors during the execution of a system call and the requested system call ended unsuccessfully. The difference between the E_IPC and this error is this error guarantees the execution result of the system call is a failure.

E_IPCS ERCD(-73, 0) Interprocessor communication status error

This error code indicates that communication between processors is not possible due to some reason.

This error code is returned when communication between processors cannot be done in normal state such as the other processor in DORMAT state or during initialization. When communication between processors is not possible in a failure state, E_IPCA is returned.

Memory Management Error Class (-257 to -260) (AMP TKSE and SMP TKSE)

E_SYSMEM ERCD(-257, 0) Insufficient system memory space

This error code is returned when there is insufficient memory space to be used inside AMP TKSE.

File Management Error Class (-261 to -280) (AMP TKSE and SMP TKSE)

E_FNAME ERCD(-261, 0) Invalid path name; invalid file name

E_FD ERCD(-262, 0) Invalid file descriptor

E_FACV ERCD(-263, 0) File access privilege error

E_PERM ERCD(-264, 0) Undeletable file

E_PWD ERCD(-265, 0) Invalid password

Should not be used by AMP TKSE.

E_ENDR ERCD(-266, 0) The end record has been reached

E_REC ERCD(-267, 0) Invalid record type

E_NOLNK ERCD(-268, 0) Not a link file

E_LOCK ERCD(-269, 0) The record is locked

E_XFS ERCD(-270, 0) Belongs to a different file system

E_NOFS ERCD(-271, 0) File system not connected

E_NODSK ERCD(-272, 0) Insufficient disk space

E_ILFMT ERCD(-273, 0) Invalid disk format

E_SEIO ERCD(-274, 0) Standard input/output error

Device Management Error Class (-281 to -290) (AMP TKSE and SMP TKSE)

E_NODEV ERCD(-281, 0) The device does not exist

E_ERDEV ERCD(-282, 0) Abnormal device status

4. AMP T-Kernel Standard Extension Functions

4.1 Memory Management Function

4.1.1 Overview of the Memory Management Function

The Memory Management Function of the “AMP TKSE” manages data memory areas. It manages four types of data memory areas: local memory area, shared memory area, system memory area and memory space between kernels. It provides the function of allocating and freeing a specified number of memory blocks for each area.

System calls provide the functions to allocate memory in blocks only. The block size depends on the MMU functions and others, and is set to the optimum value for your system.

To divide and manage the memory area smaller in size than in blocks unit, the library shall be used or the necessary processes shall be executed in your application. Normally, library shall be used instead of using system calls directly; system calls shall be directly used only if library is insufficient.

The allocated memory blocks have successive logical addresses, and the starting logical address is returned to your application. Because the logical address will not change once allocated, you can directly access the memory blocks with the returned address. You are free to write/read data to/from the allocated memory blocks, but as a general rule, programs cannot be run in them.

Because an exclusive memory access control is not provided, it shall be implemented in your application, by using semaphores, etc. if necessary.

The memory managed by the memory management function of AMP TKSE is the memory for its own processor only. It cannot operate the memory of other processors.

As for API specification of the memory management function of AMP TKSE, the operation of shared memory between kernels has been added to T-Kernel AMP TKSE Version 1.00 Specification.

4.1.2 System Calls

Get Memory Block

tkse_get_mbk

C Language Interface

```
ER ercd = tkse_get_mbk(VP *adr, INT nblk, UINT atr);
```

Parameter

| | | |
|------|------|--|
| VP | *adr | area where start address of allocated memory area is returned |
| INT | nblk | the number of allocated memory blocks (> 0) |
| UINT | atr | attributes of memory blocks [(M_COMMON M_SYSTEM · M_INTERKER)] [M_RESIDENT] [TA_DELEXIT] M_COMMON : specify common memory M_SYSTEM : specify system memory M_INTERKER : allocates the area in shared memory space between kernels M_RESIDENT : specify resident TA_DELEXIT : specify deletion on process termination |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|---|
| E_OK | normal completion |
| E_MACV | access to inaccessible access not allowed (adr) |
| E_NOMEM | insufficient memory area |
| E_SYSMEM | Insufficient system memory area |
| E_PAR | illegal parameter |

Description

Allocates the contiguous memory area as many as the number of blocks specified by “nblk” and return the start address to “*adr”.

Specify the attribute of memory area for “atr” as follows:

When “M_COMMON” attribute is specified, the memory block area is allocated to the shared memory space. This

memory block becomes accessible from all processes of its own AMP TKSE.

When “M_SYSTEM” attribute is specified, the memory areas are accessible as system memory only from systems (OS, device drivers, etc.). This specification shall not be used from application processes.

- When the M_INTERKER attribute is specified, the memory block area is allocated to the shared memory space between kernels. This memory block becomes accessible from the processes of all AMP TKSE.

When “M_COMMON”, “M_SYSTEM” and “M_INTERKER” are not specified, only local memory is enabled. Local memory is accessible only from the processes to which memory blocks are allocated.

When the “M_RESIDENT” attribute is specified, the memory area constantly exists as resident memory in the main memory without being swapped out to disks. Without the specification, it is set to nonresident memory. However, even if the M_RESIDENT attribute is not specified, the shared memory between kernels is always resident memory.

In systems without virtual memory, this setting has no particular meaning (equal to resident).

When the “TA_DELEXIT” attribute is specified, memory blocks are automatically released after the process exits which allocated the memory blocks. However, in the case of local memory, the memory blocks are released when the process is terminated with or without this setting.

Release Memory Block

tkse_rel_mbk

C Language Interface

```
ER ercd = tkse_rel_mbk(VP adr);
```

Parameter

| | | |
|----|-----|--|
| VP | adr | address of memory block to be released |
|----|-----|--|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|-------|------------------------------|
| E_OK | normal completion |
| E_PAR | illegal memory block address |

Description

Releases the memory block specified by “adr”. “adr” should be the address obtained by “tkse_get_mbk()”. The memory area allocated in the local memory space cannot be released from other processes.

Refer to Memory State

tkse_mbk_sts

C Language Interface

```
ER ercd = tkse_mbk_sts(M_STATE *pk_sts);
```

Parameter

M_STATE pk_sts area whose memory state is returned

Return Parameter

ER ercd error code

content of pk_sts

```
typedef struct m_state {
    INT      blkksz;      /* block size */
    INT      total;      /* total number of blocks */
    INT      free;       /* the number of remaining blocks */
} M_STATE;
```

blkksz memory allocation unit byte number (one block).The number of bytes in memory allocation unit(one block). In general, this is the CPU's page size. 4KB on the standard scale.

total total number of blocks across the system.

free The number of unused blocks across the system.

Error Code

E_OK normal completion

E_MACV access to inaccessible address (sts) not allowed

Description

Gets the current memory usage status and stores it in the area pk_sts displays.

The total number of blocks becomes the memory block total for all attributes currently being allocated.

Supplement

In systems with virtual memory, the total number of blocks and the number of remaining blocks may not be uniquely determined.

Therefore, concrete meanings of as implementation-dependent of each element of "pk_sts."

However, "free/total" shall be set to as a reference value to indicate the remaining memory ratio.

When concrete value cannot be set by implementation, both the total number of blocks and the number of remaining blocks may be set to-1.

4.1.3 Library Calls

Allocate Nonresident Local Memory

malloc

C Language Interface

```
void* adr = malloc(size_t size);
```

Parameter

| | | |
|--------|------|---------------------------------------|
| size_t | size | number of bytes to be allocated (> 0) |
|--------|------|---------------------------------------|

Return Parameter

| | | | |
|-------|-----|---------|--|
| void* | adr | != NULL | normal completion (allocated memory address) |
| | | = NULL | error |

Description

Allocates specified size of memory from nonresident local memory, and return the start address.

When the allocation of the memory area fails, NULL is returned.

The attribute of allocated memory is set to "TA_DELEXIT".

Allocate Nonresident Local Memory

calloc

C Language Interface

```
void* adr = calloc(size_t nelem, size_t elsize);
```

Parameter

| | | |
|--------|--------|--|
| size_t | nelem | the number of elements to be allocated (> 0) |
| size_t | elsize | the number of one element (> 0) |

Return Parameter

| | | | |
|-------|-----|---------|--|
| void* | adr | != NULL | normal completion (allocated memory address) |
| | | = NULL | error |

Description

Allocates storage area of the elements which are as many as the number of “nelem” and as large as the size of “elsize” from nonresident local memory, and return the start address.

When the allocation of the memory area fails, NULL is returned. The contents of the allocated area are initialized with 0.

The attribute of allocated memory is set to “TA_DELEXIT”.

Reallocate Nonresident Local Memory

realloc

C Language Interface

```
void* adr = realloc(void *ptr, size_t size);
```

Parameter

| | | |
|--------|------|--|
| void | *ptr | address of the area to be resized an area is newly allocated when "NULL" is specified |
| size_t | size | the number of bytes to be allocated (0) the area is released when zero is specified |

Return Parameter

| | | | |
|-------|-----|---------|--|
| void* | adr | != NULL | normal completion (allocated memory address) |
| | | = NULL | error |

Description

The size of the nonresident memory area specified by "ptr" which was already allocated to the local memory space is changed to "size" and reallocated, and the header address is returned.

If NULL is specified for ptr, the area of "size" is newly allocated to the local memory space, and the header address is returned.

If 0 is specified for "size", the area specified by "ptr" is released. At this time, "ptr" must be the address allocated by "malloc()", "calloc()", and "realloc()".

If "ptr" = NULL and size = 0 is specified at the same time, nothing is processed and NULL is returned.

If allocation of the memory area fails and release of the area is specified, NULL is returned.

"ptr" must be an address allocated by "malloc()", "calloc()", and "realloc()" in NULL or within the same process. The result when other values are specified is undefined.

Free Nonresident Local Memory

free

C Language Interface

```
void free(void *ptr);
```

Parameter

| | | |
|------|------|---------------------------------|
| void | *ptr | address of the area to be freed |
|------|------|---------------------------------|

Return Parameter

none

Description

Frees the area in nonresident local memory specified by "ptr".

If NULL is specified for "ptr", nothing is processed.

"ptr" must be the address allocated by "malloc()", "calloc()", and "realloc()" in NULL or within the same process. The result when other values are specified is undefined.

Allocate Nonresident Common Memory

Smalloc

C Language Interface

```
void* adr = Smalloc(size_t size);
```

Parameter

| | | |
|--------|------|---|
| size_t | size | the number of bytes to be allocated (> 0) |
|--------|------|---|

Return Parameter

| | | | |
|-------|-----|---------|--|
| void* | adr | != NULL | normal completion (allocated memory address) |
| | | = NULL | error |

Description

Allocates the nonresident memory area of a specified size to the shared memory space and returns the header address.

When the allocation of the memory area fails, NULL is returned.

The attribute of allocated memory is set to "TA_DELEXIT".

Allocate Nonresident Common Memory

Scalloc

C Language Interface

```
void* adr = Scalloc(size_t nelem, size_t elsize);
```

Parameter

| | | |
|--------|--------|--|
| size_t | nelem | the number of elements to be allocated (> 0) |
| size_t | elsize | the number of one element (> 0) |

Return Parameter

| | | | |
|-------|-----|---------|--|
| void* | adr | != NULL | normal completion (allocated memory address) |
| | | = NULL | error |

Description

Allocates a storage area of the elements which are as many as the number of “nelem” and as large as the size of “elsize” from nonresident common memory, and return the start address.

When the allocation of the memory area fails, NULL is returned.

The attribute of allocated memory is set to “M_COMMON”, and the area is initialized with zero.

Reallocate Nonresident Common Memory

Srealloc

C Language Interface

```
void* adr = Srealloc(void *ptr, size_t size);
```

Parameter

| | | |
|--------|------|---|
| void | *ptr | address of the area to be resized an area is newly allocated when NULL is specified |
| size_t | size | the number of bytes to be reallocated (>= 0) the area is released when zero is specified |

Return Parameter

| | | | |
|-------|-----|---------|--|
| void* | adr | != NULL | normal completion (allocated memory address) |
| | | = NULL | error |

Description

The size of the nonresident memory area specified by "ptr" which was already allocated to the shared memory space is changed to "size" and reallocated, and the header address is returned.

If NULL is specified for "ptr", the area of "size" is newly allocated to the shared memory space, and the header address is returned.

If 0 is specified for "size", the area specified by "ptr" is released. At this time, "ptr" must be the address allocated by "Smalloc()", "Scalloc()", and "Srealloc()" within the same process.

If "ptr"= NULL and size = 0 are specified at the same time, nothing is processed and NULL is returned.

If allocation of the memory area fails or the release of the area is specified, NULL is returned.

"ptr" must be NULL or an address allocated by "Smalloc()", "Scalloc()", and "Srealloc()" within the same process.

The result for when other values are specified is undefined.

Free Nonresident Common Memory

Sfree

C Language Interface

```
void Sfree(void *ptr);
```

Parameter

void *ptr address of the area to be released

Return Parameter

none

Description

Frees an area in nonresident common memory specified by "ptr".

If NULL is specified for "ptr", nothing is processed.

"ptr" must be NULL or an address allocated by "Smalloc()", "Scalloc()", and "Srealloc()" within the same process.

The result for when other values are specified is undefined.

4.2 Process/Task Management Function

4.2.1 Process/Task Management Function Overview

AMP TKSE process / task management function offers the function for carrying out parallel operation of many processes.

Process/Task management has a function about creation and termination of the process or the task, state change, and information acquisition. When performing synchronization/communication between processes and between tasks, the synchronization/communication function between tasks (event flag, message buffer, etc) and the interprocess communication function (message, global name, shared memory, etc) are used.

The process and task management function targets processes and tasks of its own AMP TKSE. It cannot operate processes and tasks of other AMP TKSE.

The API specification of the process and task management function is the same as the T-Kernel AMP TKSE Version 1.00 Specification

4.2.2 System Calls

Create/Execute Processes

tkse_cre_prc

C Language Interface

```
ER ercd = tkse_cre_prc(T_CPRC *pk_cprc, MESSAGE* msg);
```

Parameter

```
T_CPRC *pk_cprc  process creation information
MESSAGE          *msg      initial process message
```

```
typedef struct {
    ATR    pcatr;      /* process attribute */
    VP    prchr;      /* handler for the source object of a process */
    PRI    pri;        /* process priority */
                0 <= pri <= 255    any priority
                = -1                the same priority as this process
    UB    oname[8];   /* object name */

    /* other implementation-dependent information */
} T_CPRC;
```

pcatr indicates process attribute and specifies the following:

```
pcatr := (TPA_SYS || TPA_USR) | (TPA_SEIO || TPA_LINK || TPA_PTR) | TPA_ONAME
```

TPA_SYS create as a system process

TPA_USR create as a user process

TPA_SEIO a handle for the process is a path name of standard input/output file

TPA_LINK a handle for the process is a link to the file of the standard file system

TPA_PTR a handle for the process is a pointer to the codes loaded in memory

TPA_ONAME specify the object name

```
typedef struct {
    W      msg_type;    /* message type */
    W      msg_size;    /* message size (number of bytes) */
    MSGBODY msg_body;  /* message body (msg_size bytes) */
}
```

```
} MESSAGE;
```

* For details of the MSGBODY union, refer to "4.2.2 Message Structure"

Return Parameter

| | | |
|---------|------|--|
| ER ercd | >= 0 | normal completion (created process ID) |
| | < 0 | error |

Error Code

| | |
|---------|---|
| E_FACV | no access privileges (E) for the file (when TPA_SEIO, TPA_LINK is specified) |
| E_MACV | access to address (msg, hdr(TPA_PTR)) not allowed |
| E_BUSY | could not open the file because it is already opened exclusively |
| E_IO | input/output error occurred |
| E_NOEXS | file does not exist |
| E_NOFS | the file system to which the file belongs is not connected |
| E_NOMEM | insufficient memory area (insufficient memory area to load) |
| E_REC | no program record present in the file. or the content of the program record is unusual (when TPA_LINK is specified) |
| E_ONAME | the specified object name has already been used |

Description

Creates a process and allocates a process ID.

prcatr of T_CPRC structure indicates the attribute of a created process.

If TPA_SEIO attribute is specified, a new process is created using the content of the specified file as its program code. Specify the path name of the standard input/output of the target file for prchdr.

If TPA_LINK the attribute is specified, a new process is created using the content of the first executable program record in the file of the specified standard file system as its program code. Specify the link (LINK*) to the standard file system file for prchdr.

If TPA_PTR attribute is specified, new process is created using program codes in memory. Specify the pointer of the program codes in memory for prchdr. Note that the format of the program codes in memory and the running methods are implementation-dependent.

When the TPA_ONAME attribute is specified, the object name specified with oname becomes valid. When the TPA_ONAME attribute is not specified, there is no object name.

The priority of created process will be specified by pri value. At the same time, the main task is created and it starts.

When the process (main task) starts to run, the message specified by msg is passed. This message structure is essentially the same as the structure of the interprocess message.

Supplement

TPA_PTR attribute is assumed to be the romization of program codes. There are several possible ways to run the process's program codes in the ROM, such as to directly run the program in the ROM or to run it after transferring it to the RAM. The optimal method is determined according to the applications being applied and the hardware. Therefore, the format and the running method for the program codes may be determined implementation-dependently.

Exit Process

tkse_ext_prc

C Language Interface

```
void tkse_ext_prc(W code);
```

Parameter

| | | |
|---|------|-------------------|
| W | code | process exit code |
|---|------|-------------------|

Return Parameter

none

Description

Exits this process normally, and send the process normal completion message with a specified “code” to the parent process.

All the resources such as files used in invoking process are automatically released excepting certain resources (options such as “tkse_cre_sem” without “TA_DELEXIT” specification).

Terminate Other Process

tkse_ter_prc

C Language Interface

```
ER ercd = tkse_ter_prc(ID pid, W code, W opt);
```

Parameter

| | | |
|----|------|--|
| ID | pid | target process ID > 0 any process = 0 invoking process (cannot be specified: error) = -1 parent process |
| W | code | exit code |
| W | opt | specify how to terminate (TERM_NRM TERM_ALL) TERM_NRM Terminate the specified process only TERM_ALL Terminate the specified process and all the descendant processes |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|--|
| E_OK | normal completion |
| E_ILUSE | invoking process is specified (pid = 0 or PID of invoking process) E_NOEXS process (pid) does not exist |
| E_PAR | illegal parameter (specification other than "opt" = "TERM_NRM" and "TERM_ALL") |

Description

Terminates the specified process, and send the process termination message with a specified "code" to the parent process of the specified process.

When "TERM_ALL" is specified, specified process and all the descendant processes are killed. In this case, the termination messages of the descendant processes of the specified process are not sent. When the parent process of invoking process or further its own parent process is specified, invoking process is also killed.

Change Priority of Processes/Tasks

tkse_chg_pri

C Language Interface

```
ER ercd = tkse_chg_pri(ID id, PRI pri, W opt);
```

Parameter

| | | |
|-----|-----|--|
| ID | id | target process ID or task ID |
| PRI | pri | priority to be changed |
| W | opt | specify how to change the priority (P_ABS P_REL) [P_TASK] P_ABS absolute specification (change to specified priority) P_REL relative specification (change to current priority + "pri") P_TASK Set task as target |

Return Parameter

| | | | |
|----|------|------|--|
| ER | ercd | >= 0 | normal completion (priority after change: 0-255) P_TASK specified priority of task with ID id P_TASK unspecified priority of main task of a process with ID id |
| | | < 0 | error code |

Error Code

| | |
|---------|---|
| E_NOEXS | process (id) does not exist |
| E_ID | no task (id) existent or, no task in invoking process |
| E_PAR | priority value is out of range (in relative specification, new priority is out of current priority group) illegal parameter (specification of the parameter other than opt = "P_ABS", "P_REL", or "P_TASK") |

Description

Changes the priority of the specified process/task.

When "P_TASK" is not specified:

| | |
|---------|---|
| id = 0 | Change the priorities of all the tasks in invoking process. |
| id = -1 | Change the priorities of all the tasks in the parent process. |

id > 0 Change the priorities of all the tasks in a process with process ID specified by "id".

When "P_TASK" is specified:

id = 0 Change the priority of invoking task.

id > 0 Change the priority of task with task ID specified by "id". The tasks which can be specified are only the tasks in invoking process.

When "P_ABS" is specified (absolute specification), the priority after change is set to the value specified by "pri".

When "P_REL" is specified (relative specification), the priority after change is set to current priority value added by the value specified by "pri".

In a priority change with relative specification, a priority cannot be changed to the priority other than the priorities in current priority group.

Get Process State

tkse_prc_sts

C Language Interface

```
ER ercd = tkse_prc_sts(ID pid, P_STATE* buff, TC* name);
```

Parameter

| | | |
|----------|------|--|
| ID | pid | target process ID |
| | > 0 | any process |
| | = 0 | invoking process |
| | = -1 | parent process |
| P_STATE* | buff | storage area in process state (not stored in the case of NULL) |
| TC* | name | storage area of process name (area for process name's maximum length + one character) (not stored in the case of NULL) |

Return Parameter

| | | | |
|----|------|-----|--|
| ER | ercd | > 0 | normal completion (specified process ID) |
| | | < 0 | error code |

P_STATE* buff process state

```
typedef struct {
    UW    state;        /* process state */
    PRI   priority;    /* current process priority (0 - 255) */
    ID    parpid;      /* process ID of the parent process */
} P_STATE;
```

The content of name Process name

Error Code

| | |
|---------|---|
| E_MACV | access to inaccessible address (buff, path) not allowed |
| E_NOEXS | process (pid) does not exist |

Description

Retrieves the process state specified by “pid”, and store it in the area specified by “buff”. Also the process name of the specified process shall be stored in the area specified by “name”. When either “buff” or “name” is set to NULL, no information is stored.

The process name is a name added by the system when the process is created. When the process is created from the file of a standard file system, the file name becomes the process name. In other cases, the name created by the system automatically becomes the process name.

The process state (state) is as follows: Each value "1" indicates that a process is in the state.

MSB

LSB

reserved

reserved

P_WAIT wait state

P_READY ready state

P_RUN run state

[Figure 8] Process state

Get Statistics Information about Processes

tkse_get_inf

C Language Interface

```
ER ercd = tkse_get_inf(ID pid, P_INFO* buff);
```

Parameter

| | | |
|--------------|------|---|
| ID | pid | target process ID |
| | > 0 | any process |
| | = 0 | invoking process |
| | = -1 | parent process |
| P_INFO* buff | | storage area of statistical information |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

The content of buffer

```
typedef struct {
    UW    etime;    /*cumulative elapsed time (in seconds)*/
    UW    utime;    /*cumulative CPU time spent in process*/
    UW    stime;    /*cumulative CPU time spent in system*/
    W     tmem;     /*total memory size required to execute*/
    W     wmem;     /*currently allocated actual memory size*/
    W     resv[11]; /*reserved*/
} P_INFO;
```

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | normal completion |
| E_MACV | access to inaccessible address (buff) not allowed |
| E_NOEXS | process (pid) does not exist |

Description

Gets statistical information on the specified process.

“utime” and “stime” are set to the total time of all tasks existing at the time which are included in the process. Therefore, the time spent by previously terminated tasks is not included. The sum of “utime” and “stime” is the cumulative CPU time spent by the process.

Process Exit Message

tkse_req_emg

C Language Interface

```
ER ercd = tkse_req_emg(ID pid, W t_mask);
```

Parameter

| | | |
|----|----------|---|
| ID | pid | target process ID |
| | | = -1 parent process |
| | | > 0 any process |
| W | t_mask | specify exit message type (specify with OR) |
| | 0 | clear notification |
| | MM_ABORT | notify when target process is aborted |
| | MM_EXIT | notify when target process is terminated normally |
| | MM_TERM | notify when target process is killed |

Return Parameter

| | | | |
|----|------|-----|---------------------------------------|
| ER | ercd | > 0 | normal completion (original "t_mask") |
| | | < 0 | error code |

Error Code

| | |
|---------|--|
| E_ILUSE | invoking process is specified ("pid = 0" or "PID" of invoking process) |
| E_NOEXS | process (pid) does not exist |
| E_PAR | illegal parameter (illegal t_mask) |

Description

Sending of the exit message shall be set to invoking process when the process specified by "pid" is terminated. "pid= -1" indicates a parent process. Invoking process cannot be specified (E_ILUSE).

Specify the types of termination to be notified by "t_mask".

```
t_mask = [MM_ABORT] | [MM_EXIT] | [MM_TERM]
```

| | |
|----------|---|
| MM_ABORT | notify when target process is aborted |
| MM_EXIT | notify when target process normally exits |

MM_TERM notify when target process is terminated

When “t_mask = 0”, exit message is cleared. When “t_mask < 0”, the setting remains unchanged.

The old setting value of “t_mask” shall be returned as a return value when invoking process is terminated, the setting is automatically cleared.

The exit messages have the following formats:

```
typedef struct {
    W      type;      /* message type (MS_SYS2) */
    W      size;      /* message size */
    W      kind;      /* termination type (MS_ABORT,MS_EXIT,MS_TERM) */
    ID     pid;       /* process ID of the terminated process */
    W      code;      /* exit code */
} EXITMSG;
```

“kind”, “pid”, and “code” have the same content as the termination message to be sent to parent process.

```
kind      termination message type (any of “MS_ABORT”, “MS_EXIT”, or “MS_TERM”)
pid       process ID of the terminated process
code      system error code or exit code specified by “tkse_ext_prc()” and “tkse_tE_prc()”
```

“EXITMSG” is set to one of various “MS_SYS2” system messages.

```
typedef union {
    struct {          /* MS_SYS2 basic form*/
        W      type;      /* message type (MS_SYS2) */
        W      size;      /* message size */
        W      kind;      /* kind */
        VW     info[1];   /* various information different per each type */
    } base;
    EXITMSG exitmsg; /* exit message*/
} MSG_SYS2;
```

The exit message is sent differently from normal completion messages notifying the parent process when child process is terminated. Therefore, if exit message is set to a child process, a further termination message may be received after receiving the exit message (MS_ABORT,MS_EXIT,MS_TERM)..

Retrieve Various Information about Processes

tkse_prc_inf

C Language Interface

```
ER ercd = tkse_prc_inf(ID pid, W item, VP buf, W len);
```

Parameter

| | | |
|----|------|---|
| ID | pid | target process ID = 0 this process = -1 parent process > 0 any process |
| W | item | type of information PI_LINK (0x00010000) retrieve the link to program file PI_NTSK (0x00020000) retrieve the number of tasks in a process PI_TSKSTAT (0x00030000) retrieve the states of each task PI_CREINF (0x00040000) information during process creation |
| VP | buf | buffer for storing information (If NULL, not stored) |
| W | len | byte length of buffer area (buf) for storing information |

Return Parameter

| | | |
|----|------|--|
| ER | ercd | > 0 normal completion (size necessary for buf (number of bytes)) < 0 error code |
|----|------|--|

The content of buffer Various information on the acquired process

Error Code

| | |
|---------|--|
| E_MACV | access to address (buff, path) not allowed |
| E_NOEXS | process (pid) does not exist |
| E_PAR | illegal parameter (insufficient len, illegal item) |

Description

Retrieves various information about the process specified by pid (process ID) to store in buf.

Specify the type of information as an item. pid = 0 indicates this process and pid = -1 indicates parent process.

len indicates buf size (number of bytes). If len is less than the necessary size, an error (E_PAR) occurs and nothing

is stored in buf. Returns the size (number of bytes) necessary for buf. If buf = NULL is specified, information will not be stored, but the size necessary for buf will be returned as a return value. In this case, len will be ignored.

Specify one of the following for the type of information (item):

```
#define PI_LINK          0x00010000    /* link to the program file */
#define PI_NTSK         0x00020000    /* number of tasks in a process */
#define PI_TSKSTAT      0x00030000    /* each task state */
#define PI_CREINF       0x00040000    /* information during process creation */
```

PI_LINK :

```
item    PI_LINK
buf     LINK    Link information for the program file
```

Retrieves the link to program file.

Information can be acquired only if the process is created from a link for a standard file system.

PI_NTSK :

```
item    PI_NTSK
buf     W ntsk
```

Retrieves the number of tasks (total number of main and sub tasks) in the processes.

PI_TSKSTAT :

```
item    PI_TSKSTAT + n
buf     P_TSKSTAT tskstat  Task status information
```

typedef struct {

```
        ID  tskid;    /* task ID */
        UW  state;    /* task state */
        PRI  priority; /* task priority */
```

```
    } P_TSKSTAT;
```

```
state := P_DORMANT || P_WAIT || P_READY || P_RUN
```

Retrieves the state information about the nth task. If n = 0, then main task will be retrieved. If n >= 1, then subtask will be retrieved. n is valid only until number of tasks retrieved by PI_NTSK is minus 1.

PI_CREINF :

```
item    PI_CREINF
buf     P_CREINF creinf    Process creation information
```

typedef struc {


```
        PRI  pri;          /* process priority */
        ATR  prcatr;       /* process attribute */
        VB   prchdr[1];   /* handler for the source object of a process */
    } P_CREINF;
```

Returns information during process creation with `tkse_cre_prc()`.

Since the size of `prchdr[1]` is not a fixed length, it is necessary to acquire actual information after acquiring the size of the area necessary for storing all information and then securing buf.

Subtask Creation

tkse_cre_tsk

C Language Interface

```
ER ercd = tkse_cre_tsk(FP entry, PRI pri);
```

Parameter

| | | |
|----|-------|-----------------------|
| FP | entry | subtask start address |
| W | pri | subtask priority |

Return Parameter

| | | | |
|----|------|-----|--|
| ER | ercd | > 0 | normal completion (created subtask ID) |
| | | < 0 | error code |

Error Code

| | |
|---------|------------------------------|
| E_MACV | illegal address (entry) |
| E_LIMIT | subtask count limit exceeded |
| E_NOMEM | insufficient memory area |

Description

Creates a subtask in this process. The created subtask enters dormant state.

The task ID of the created subtask is returned if creation is successful.

The subtask is defined as a function in the following format:

```
void subtask(W arg)
{
    /* Program execution code */
    tkse_ext_tsk();
}
```

As for the argument arg of the subtask function, the subtask start parameter specified by subtask start tkse_sta_tsk() is passed.

When subtask ends, tkse_ext_tsk() is used. Subtask cannot end in a return.

Subtask Startup

tkse_sta_tsk

C Language Interface

```
ER tkse_sta_tsk(ID id, W arg)
```

Parameter

| | | |
|----|-----|-------------------------|
| ID | id | subtask ID |
| W | arg | subtask start parameter |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | normal completion |
| E_ID | illegal task ID (id is invalid or cannot be used) |
| E_NOEXS | object does not exist (the task specified in id does not exist) |
| E_OBJ | illegal object state (the task is not in DORMANT state) |

Description

Runs the subtask created by `tkse_cre_tsk()`.

Only a subtask in dormant state can be started. If a task in another state is about to start, error code `E_OBJ` is returned.

Subtask Creation and Startup

tkse_crs_tsk

C Language Interface

```
ER ercd = tkse_crs_tsk(FP entry, PRI pri, W arg);
```

Parameter

| | | |
|----|-------|-------------------------|
| FP | entry | subtask start address |
| W | pri | subtask priority |
| W | arg | subtask start parameter |

Return Parameter

| | | | |
|----|------|-----|--|
| ER | ercd | > 0 | normal completion (created subtask ID) |
| | | < 0 | error code |

Error Code

| | |
|---------|------------------------------|
| E_MACV | illegal address (entry) |
| E_LIMIT | subtask count limit exceeded |
| E_NOMEM | insufficient memory area |

Description

Creates and starts a subtask in this process.

The task ID of the created subtask is returned when creation is successful.

Equivalent to `tkse_cre_tsk()` call excepting that it enters into executable state after creation.

Exit invoking task

tkse_ext_tsk

C Language Interface

```
void    tkse_ext_tsk(void);
```

Parameter

none

Return Parameter

none

Description

Exits invoking task.

It can also be used from either the main task or the subtask.

When exiting the main task, process is exited. Therefore, all tasks in the process will be exited.

Terminate Other Task

tkse_ter_tsk

C Language Interface

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

Parameter

| | | |
|----|-------|----------------|
| ID | tskid | target task ID |
|----|-------|----------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------|-------------------------|
| E_OK | normal completion |
| E_ID | illegal task ID (tskid) |

Description

Terminates the specified task.

Only subtasks in invoking process can be specified. Invoking task and the main task cannot be specified.

Task Sleep

tkse_slp_tsk

C Language Interface

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

Parameter

| | | |
|-----|-------|---|
| TMO | tmout | timeout period |
| | > 0 | wait for a specific period of time (milliseconds) |
| | = -1 | wait for an infinite period of time |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | normal completion |
| E_TMOUT | not woken up although the timeout period has expired |
| E_DISWAI | waiting suspended because message handler is invoked |
| E_PAR | illegal parameter (time) |

Description

Puts this task into sleep state.

Prior to when the timeout period specified by tmout expires, if wakeup is conducted by tkse_wup_tsk in this task from another task, the WAIT state is released and normal completion E_OK is returned.

When wakeup by tkse_wup_tsk is not conducted while the timeout period specified by tmout expires, the WAIT state is released and timeout error code E_TMOUT is returned.

If the message is received during the period when the task is in WAIT state and the message handler starts, the WAIT state is released and error E_DISWAI is returned.

Task Wake up

tkse_wup_tsk

C Language Interface

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

Parameter

| | | |
|----|-------|----------------|
| ID | tskid | target task ID |
|----|-------|----------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|--|
| E_OK | normal completion |
| E_ID | illegal task ID (tskid) |
| E_LIMIT | exceeded the limit of wake-up request queueing |

Description

When the task specified by “tskid” is in the sleep state, the wait state is released. When the specified task is not in the sleep state, the wake-up request is queued.

Only tasks in invoking process can be specified. Tasks of other process cannot be woken up. Besides, invoking task cannot be specified.

Cancel Task Wake-up Request

tkse_can_wup

C Language Interface

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

Parameter

| | | |
|----|-------|-------------------|
| ID | tskid | target task ID |
| | | > 0 any task |
| | | = 0 invoking task |

Return Parameter

| | | |
|----|------|---|
| ER | ercd | >= 0 Normal completion (Number of queued wakeup requests) |
| | | < 0 error code |

Error Code

| | |
|------|---------------------------|
| E_ID | Task ID(tskid) is invalid |
|------|---------------------------|

Description

Cancels all queued wakeup requests in tasks specified by tskid, and returns the canceled number of queuing wakeup requests.

When completion is normal, the number of queued wakeup requests is returned.

Only tasks in invoking process can be specified.

Delay Task

tkse_dly_tsk

C Language Interface

```
ER ercd = tkse_dly_tsk(RELTIM dlytim);
```

Parameter

RELTIM dlytim delay time (milliseconds >= 0)

Return Parameter

ER ercd error code

Error Code

| | |
|----------|--|
| E_OK | normal completion |
| E_DISWAI | waiting suspended because message handler is invoked |
| E_PAR | illegal parameter (dlytim) |

Description

Puts invoking task into the wait state for specified time duration.

When a message is received during the period when the task is in WAIT state and the message handler starts, the WAIT state is released and error code E_DISWAI is returned.

Unlike tkse_slp_tsk(), when the delay time specified by dlytim expires, normal completion E_OK is returned. In addition, even if the wakeup request is conducted by tkse_wup_tsk() during delay time, it does not become wait release.

When "dlytim = 0" is specified, the execution is suspended and the task is rescheduled. That is, this changes the task from the execution state to the executable state.

Get Invoking Task ID

tkse_get_tid

C Language Interface

```
ER ercd = tkse_get_tid();
```

Parameter

none

Return Parameter

| | | | |
|----|------|-----|--------------------------------------|
| ER | ercd | > 0 | normal completion (invoking task ID) |
| | | < 0 | error code |

Description

Gets task ID of invoking task.

Load Module

tkse_lod_mod

C Language Interface

```
ER ercd = tkse_lod_mod(T_LMOD *pk_mod, P_DYNLDINF *info);
```

Parameter

T_LMOD *pk_mod load module information
 P_DYNLDINF* info information concerning loaded object

```
typedef struct {
    ATR      modatr;      /* load module attribute */
    VP      modhdr;      /* handler for a load module */
} T_LMOD;
```

modatr indicates an attribute of the load module and is specified as follows:

```
modatr := (TMA_SEIO || TMA_LINK || TMA_PTR)
```

TMA_SEIO a handle for the load module is a standard input/output file path
 TMA_LINK a handle for the load module is a link to the file of the standard file system
 TMA_PTR a handle for the load module is a pointer to the codes loaded in memory

```
typedef struct {
    VP      loadaddr;      /* load address */
    UW      loadsize;      /* load size */
    FP      entry;          /* entry address */
    UW      info[3];        /* machine-dependent information */
} P_DYNLDINF;
```

Return Parameter

| | | | |
|----|------|-----|-----------------------------|
| ER | ercd | > 0 | normal completion (load ID) |
| | | < 0 | error code |

Error Code

| | |
|---------|---|
| E_FACV | no access privileges (E) for the file (when TMA_SEIO, TMA_LINK is specified) |
| E_MACV | access to address (info, hdr(TMA_PTR)) not allowed |
| E_BUSY | could not open the file because it is already opened exclusively |
| E_IO | input/output error occurred |
| E_NOEXS | file does not exist |
| E_NOFS | the file system to which the file belongs is not connected |
| E_NOMEM | insufficient memory area (insufficient memory area to load) |
| E_REC | no program record existent in the file. Or the content of program record is unusual (when TMA_LINK is specified). |

Description

Loads a load module into the local space of this process, and then allocates a load ID to it.

modatr of T_LMOD structure indicates an attribute of the load module.

If TMA_SEIO attribute is specified, the content of the specified file is loaded as a load module. Specify the path name of the standard input/output of the target file for modhdr.

If TMA_LINK the attribute is specified, the content of the first executable program record in the file of the specified standard file system is loaded as load modules. Specify the link (LINK*) to the standard file system file for modhdr.

If TMA_PTR attribute is specified, object code in memory may be loaded as a load module. Specify the pointer of the object codes in memory for modhdr. Note that the format of the object codes in memory and the running methods are implementation-dependent.

If the load is successful, returns information concerning the loaded load module to info.

The load module is loaded (mapped) in memory, but no processing such as a relocation is done. If the same load module as that has been already loaded is also specified, another new memory space is allocated to load it.

Only the load module is simply loaded on memory (mapping), and processing of the relocation of the symbol address, etc. is not conducted. In addition, even when the same module as the load module that has already been loaded is specified, another memory space is newly allocated and loading is conducted. In this case, separate load IDs are respectively allocated.

Unload Module

tkse_unl_mod

C Language Interface

```
ER ercd = tkse_unl_mod(ID loadid);
```

Parameter

| | | |
|----|--------|--|
| ID | loadid | load ID of load module (ID obtained by “tkse_lod_mod()”) |
|----|--------|--|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------|--------------------------------------|
| E_OK | normal completion |
| E_ID | specified load module does not exist |

Description

Unloads the load module specified by “loaded”.

For the area mapped on memory for the load module, all maps are released.

There is no concern whether the load module is in use or not.

Meanwhile, all load modules are automatically unloaded when the process is terminated.

4.3 Interprocess Message Function

4.3.1 Interprocess Message Function Overview

The interprocess messages function of the “AMP TKSE” provides the functions to send and receive messages among any processes.

Each process has a specific message queue, and interprocess message communication is executed through this message queue. The destination of the message is specified by the process ID. The source is also discriminated by the process ID of sending process.

This function is also used for sending exit messages of the child process by the system as well as communicating interprocesses message by application.

The interprocess message can be sent to this process as well as other processes. As an example of this type of message, there are timeout messages.

The message queue is FIFO and messages are always put in the order of sending. When the message queue on the receiving side is full at sending of messages, you can designate a wait until availability of queue or a return on error.

Normally, received messages are stored in the message queue and picked up by the request to receive message. However, definition of message handler allows an asynchronous processing of message when messages of the specified type are received.

In the message function between processes, the sending and receiving of messages to and from the processes of its own AMP TKSE as well as the processes of other AMP TKSE is possible.

The API specification of the message function between processes is the same as the T-Kernel AMP TKSE Version 1.00 Specification

4.3.2 Message Type

Messages are classified into 31 types of messages from 1 to 31 according to the type number.

The message type is defined as follows:

```

#define MS_ABORT      ( 1)    /* abort process */
#define MS_EXIT       ( 2)    /* exit process */
#define MS_TERM       ( 3)    /* terminate process */
#define MS_TMOUT      ( 4)    /* timeout */
#define MS_SYSEVT     ( 5)    /* system event (kill) */
#define MS_SYS1       ( 6)    /* used in system */
#define MS_SYS2       ( 7)    /* used in system */
#define MS_SYS3       ( 8)    /* used in system */
#define MS_SYS4       ( 9)    /* used in system */
#define MS_SYS5       (10)    /* used in system */
#define MS_MNG0       (11)    /* reserved */
#define MS_MNG1       (12)    /* reserved */
#define MS_MNG2       (13)    /* reserved */
#define MS_MNG3       (14)    /* reserved */
#define MS_MNG4       (15)    /* reserved */
#define MS_MNG5       (16)    /* reserved */
#define MS_MNG6       (17)    /* reserved */
#define MS_MNG7       (18)    /* reserved */
#define MS_MNG8       (19)    /* reserved */
#define MS_MNG9       (20)    /* reserved */
#define MS_MNG10      (21)    /* reserved */
#define MS_MNG11      (22)    /* reserved */
#define MS_MNG12      (23)    /* reserved */
#define MS_TYPE0      (24)    /* application definition */
#define MS_TYPE1      (25)    /* application definition */
#define MS_TYPE2      (26)    /* application definition */
#define MS_TYPE3      (27)    /* application definition */
#define MS_TYPE4      (28)    /* application definition */
#define MS_TYPE5      (29)    /* application definition */
#define MS_TYPE6      (30)    /* application definition */
#define MS_TYPE7      (31)    /* application definition */

#define MS_MIN        ( 1)    /* minimal message type */
#define MS_MAX        (31)    /* maximal message type */

```

Message type number 0 is used inside the AMP TKSE system in order to receive and pass the start message when a process is created. This number cannot be directly used by the message function between processes.

Each message type is associated with a bit ready type mask, and multiple target message types can be specified

by using a union (OR) pattern of type masks.

The message type mask is defined as follows:

```

#define MSGMASK(msgtype)      (1 << ((msgtype) - 1))

#define MM_ABORT      MSGMASK(MS_ABORT)    /* abort process */
#define MM_EXIT       MSGMASK(MS_EXIT)     /* exit process */
#define MM_TERM       MSGMASK(MS_TERM)     /* terminate process */
#define MM_TMOU       MSGMASK(MS_TMOU)     /* timeout */
#define MM_SYSEVT     MSGMASK(MS_SYSEVT)   /* system event (kill) */
#define MM_SYS1       MSGMASK(MS_SYS1)     /* used in system */
#define MM_SYS2       MSGMASK(MS_SYS2)     /* used in system */
#define MM_SYS3       MSGMASK(MS_SYS3)     /* used in system */
#define MM_SYS4       MSGMASK(MS_SYS4)     /* used in system */
#define MM_SYS5       MSGMASK(MS_SYS5)     /* used in system */
#define MM_MNG0       MSGMASK(MS_MNG0)     /* reserved */
#define MM_MNG1       MSGMASK(MS_MNG1)     /* reserved */
#define MM_MNG2       MSGMASK(MS_MNG2)     /* reserved */
#define MM_MNG3       MSGMASK(MS_MNG3)     /* reserved */
#define MM_MNG4       MSGMASK(MS_MNG4)     /* reserved */
#define MM_MNG5       MSGMASK(MS_MNG5)     /* reserved */
#define MM_MNG6       MSGMASK(MS_MNG6)     /* reserved */
#define MM_MNG7       MSGMASK(MS_MNG7)     /* reserved */
#define MM_MNG8       MSGMASK(MS_MNG8)     /* reserved */
#define MM_MNG9       MSGMASK(MS_MNG9)     /* reserved */
#define MM_MNG10      MSGMASK(MS_MNG10)    /* reserved */
#define MM_MNG11      MSGMASK(MS_MNG11)    /* reserved */
#define MM_MNG12      MSGMASK(MS_MNG12)    /* reserved */

#define MM_TYPE0      MSGMASK(MS_TYPE0)    /* application definition */
#define MM_TYPE1      MSGMASK(MS_TYPE1)    /* application definition */
#define MM_TYPE2      MSGMASK(MS_TYPE2)    /* application definition */
#define MM_TYPE3      MSGMASK(MS_TYPE3)    /* application definition */
#define MM_TYPE4      MSGMASK(MS_TYPE4)    /* application definition */
#define MM_TYPE5      MSGMASK(MS_TYPE5)    /* application definition */
#define MM_TYPE6      MSGMASK(MS_TYPE6)    /* application definition */
#define MM_TYPE7      MSGMASK(MS_TYPE7)    /* application definition */

#define MM_ALL        (0x7fffffff)         /* all masks */
#define MM_NULL       (0)                  /* blank mask */

```

4.3.3 Message Structure

A message has the following structure:

```
typedef struct message {
    W        msg_type;        /* message type */
    W        msg_size;        /* message body size (bytes) */
    MSGBODY msg_body;        /* message body */
} MESSAGE;
```

The structure of the message body (MESSAGE) is determined by “msg_type”.

4.3.4 System Message

The Messages with message number 1-5 are called system messages. These are messages to notify the application about events that occurred in the system. Meanwhile, applications are not particularly prohibited to transfer system messages

The system messages are essentially not affected by an overflow of message queing and would not be discarded. For this reason, it is necessary for processes that receive system messages to discard system messages placed in the message queue by receiving the messages.

Each system message is described as follows:

(1) MS_ABORT -- abort message of the child process

It is automatically sent from child process to parent process when a process is aborted by a system error.

```
W        msg_type : 1  message type = MS_ABORT
W        msg_size : 8  the number of message body bytes (8byte)
MSGBODY msg_body :   message body
                struct {
                    ID pid;    /* process ID of the terminated child process */
                    W  code;    /* system error code */
                };
```

Where a code is a generated system error code, and is set to zero for an abort with “MH_TERM” message handler.

(2) MS_EXIT -- normal completion message of the child process

It is automatically sent from child process to parent process when a process is normally terminated by the system call “tkse_ext_prc()”.

```

W          msg_type : 2  message type = MS_EXIT
W          msg_size : 8  the number of message body bytes (8byte)
MSGBODY msg_body :   message body
                struct {
                    ID  pid;    /* process ID of the terminated child process */
                    W   code;   /* exit code specified by "tkse_ext_prc()" */
                };

```

(3) MS_TERM -- termination message of the child process

It is automatically sent from child process to parent process when a process is terminated by the system call "tkse_ter_prc()".

```

W          msg_type : 3  message type = MS_EXIT
W          msg_size : 8  the number of message body bytes (8byte)
MSGBODY msg_body :   message body
                struct {
                    ID  pid;    /* process ID of the terminated child process */
                    W   code;   /* exit code specified by "tkse_ter_prc()" */
                };

```

(4) MS_TMOUT -- timeout message of invoking process

A timeout message requested by the system call "tkse_req_tmng()". It is automatically sent to this process after the specified time period.

```

W          msg_type : 4  message type = MS_TMOUT
W          msg_size : 4  the number of message body bytes (8byte)
MSGBODY msg_body :   message body
                struct {
                    W   code;   /* code specified by tkse_req_tmng() */
                };

```

4.3.5 Message Handler

The message handler is a mechanism to process the reception of interprocess messages asynchronously.

The message handler processes the messages asynchronously to the ongoing process when messages of the specified type are received. Therefore, up to 31 types of message handlers corresponding to the respective message types can be simultaneously defined.

The message handlers are executed as follows:

- When a message with the corresponding type is received, the message handler is executed to interrupt the current process in the main task.
- Also when a message is sent in the wait state by some system call, the message handler is invoked, and the main task of the process goes into an execution state. In this case, a system call whose main task is in a long wait state is unconditionally interrupted, and is led to an uncertain result. That is, when message handler is terminated, the interrupted system call returns as “E_DISWAI” error code.
- The message handler works as a part of the normal process codes, there is no limit on the executable system calls, etc.
- When the message handler is terminated, the main task is usually resumed from the interrupted point; but, it is possible to move the execution point to any position of the main task (position specified by “setjmp”) using “longjmp ()” directly from the message handler.
- The message handler is not nested. More specifically, a start-up of new message handler (including other message types) waits until the currently processing message handler exits.
- When the message handler is invoked, the message which invoked the message handler is picked up from the message queue, and the pointer is passed as the parameter of a handler.
- The message handler must always exits by “tkse_ret_msg()” system call.

The message handlers are defined as functions with the following form:

```
void msg_hdr(W pid, MESSAGE *r_msg)
{
/*      where “pid” is a sending process ID. (zero for this process) */
/*      “r_msg” is a pointer to the received message. */

    Process of Received Messages

    tkse_ret_msg (0); /* exit (when returning to the interrupted point) */

    or

    tkse_ret_msg (1); /* exit (when moving to any point) */
    longjmp (reent, code); /* jump to reent */
}
```

You can use the following message handlers defined as special message handlers by system instead of message

handler function address:

(1) MH_NONE : Messages are ignored without any processing. In this case, system calls with a wait are exceptionally not interrupted. This is used to simply ignore the specific types of messages entirely without queueing.

(2) MH_BREAK : Messages are ignored without any processing. In this case, system calls with a wait are not interrupted, and “E_DISWAI” error code is returned. This is used to process timeouts, etc.

(3) MH_TERM : invoking process are aborted and “MS_ABORT” message (error code = 0) are sent to parent process.

The initial process cannot use message handlers other than MH_NONE. When a message handler other than MH_NONE is specified, the handler is not executed even if the message is received. Moreover, the specifications for MH_BREAK and MH_TERM are also ignored.

4.3.6 System Calls

Send Message

tkse_snd_msg

C Language Interface

```
ER ercd = tkse_snd_msg(ID pid, MESSAGE* msg, W opt);
```

Parameter

| | | |
|----------|-----|---|
| ID | pid | destination process ID > 0 any process = 0 invoking process = -1 parent process |
| MESSAGE* | msg | sending message |
| W | opt | specify how to wait for sending (NOWAIT WAIT CONFM) NOWAIT : not wait for message queue to be free WAIT : wait for message queue to be free CONFM: wait to receive messages |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|---|
| E_OK | normal completion |
| E_MACV | access to inaccessible address (msg) not allowed |
| E_DISWAI | wait processing interrupted because message handler is invoked |
| E_NOEXS | process (pid) does not exist |
| E_PAR | illegal parameter (illegal option and illegal message type) |
| E_ILUSE | invoking process is specified ("pid = 0" or "PID of invoking process")(when "CONFM" is specified) |
| E_LIMIT | size of message body exceeded the system limit, or is zero or less |
| E_SYSMEM | insufficient system memory area (destination message queue is full (when "NOWAIT" is specified)) |

Description

Sends messages to the process specified by "pid".

"opt" specifies the behavior of send wait when the message is transmitted.

In the case "opt = NOWAIT" is specified, the task is normally terminated when messages are put in the message queue of the destination process. When the destination message queue is full, an exit by error occurs.

In the case "opt = WAIT" is specified, the task is normally terminated when messages are put in the message queue of the destination process. When the message queue is full, it waits for it to be free. When the destination process is terminated during a waiting, an exit by error occurs.

A message is put in the message queue of the destination process in the case "opt = CONFM" is specified, and then the task is normally terminated when the message sent by the destination process is received or when it is cleared from the message queue. Wait until then. When only header section is obtained by "tkse_rcv_msg()" with the "CHECK" option, the message is not considered to be received. When a message is received with an "NOCLR" option, it is considered to be received even if it remains in the queue. When the destination process is terminated during a waiting, an error exit occurs. When this process is the destination, the "CONFM" option causes an error.

The destination process of a message can specify the processes of other AMP TKSE. In other words, messages between processes can be sent between AMP TKSE. However, if the destination process is the process of another AMP TKSE, opt = WAIT cannot be specified

Receive Message

tkse_rcv_msg

C Language Interface

```
ER ercd = tkse_rcv_msg(W t_mask, MESSAGE* msg, W msgsz, W opt);
```

Parameter

| | | |
|----------|--------|--|
| W | t_mask | message type mask targeted to be received |
| MESSAGE* | msg | storage area of received message |
| W | msgsz | byte size of total storage area of received message. "msgsz >= 8" is required since message header section is included. |
| W | opt | specify the action to receive (WAIT NOWAIT WAIEVT) (CLR NOCLR) (CHECK) WAIT : wait to receive messages of the specified type NOWAIT : not wait for messages of the specified type WAIEVT : wait for the messages of the specified type to be received and the event to occur CLR : after receiving message, the message is eliminated from the queue NOCLR : after receiving message, the message is left in the queue CHECK : check whether there are messages or not |

Return Parameter

| | | | |
|----|------|-----|--|
| ER | ercd | > 0 | normal completion (source process ID of received message) |
| | | = 0 | normal completion (invoking process is the source of received message) |
| | | < 0 | error code |

Error Code

| | |
|----------|--|
| E_MACV | access to inaccessible address (msg) not allowed |
| E_DISWAI | wait processing interrupted because message handler is invoked |
| E_TMOUT | no messages of the specified type are existent (t_mask) (When NOWAIT is specified) |
| E_PAR | illegal parameter ("msgsz" is too small. "t_mask<=0" When non- "WAIEVT" is specified, "t_mask<0" When "WAIEVT" is specified) |

Description

Receives the messages of the specified type to invoking process.

When received messages cannot be put in specified area, they are stored in the area of "msg" only within the range of "msgsz" and an error exit occurs. In this case, messages are left in the buffer regardless of the CLR option. However, when "msgsz < 8", nothing is stored in the area of "msg" and return the errors. When the entire message cannot be stored, the actual message size is determined from the header section of the stored messages. Therefore, this system call will be executed again after preparing sufficient size of area to store the message. "opt" specifies the behavior when the message is received.

In the case "opt = WAIT" is specified, wait until the arrival of message when the messages of the specified type have not been received.

In the case "opt = NOWAIT" is specified, execute an error exit when the messages of the specified type have not been received.

In the case "opt = WAIEVT" is specified, the basic behavior of the "WAIEVT" is the same as the "WAIT", but the notification of event occurrence by "tkse_brk_msg()" clears the wait state even if messages have not been received. In this case, an error (E_NOME) exit occurs.

In the case "WAIEVT" is specified, "t_mask = 0" can also be specified. In this case, no message will be received and the task waits until the notification of event occurrence.

Only one "WAIEVT" can be simultaneously specified for tasks over the entire system, When multiple tasks invoke "tkse_rcv_msg()" with the "WAIEVT" option, only the "WAIEVT" of last invoked task is available. Other tasks are processed in the same manner as the "WAIT" option.

"WAIEVT" is premised on using in the upper system (T-Shell, etc) than AMP TKSE. "WAIEVT" usually shall not be specified by applications.

In the case "opt = CLR" is specified, eliminate the messages from the queue after the messages are received.

In the case opt = "NOCLR" is specified, leave the messages in the queue even after the messages are received.

In the case "opt = CHECK" is specified, the behavior is as follows:

1. When no message exists, the "WAIT" or "WAIEVT" puts the task in a wait state, and the "NOWAIT" causes error exit.
2. When messages of the specified type exist, the task is normally terminated after the messages are stored in "msg". The "CLR" eliminates messages from the queue while the "NOCLR" leaves messages in the queue.
3. When messages of the specified type do not exist and the other types of messages exist, only the top 8 bytes of the messages (msg_type and msg_size) are stored in "msg" and the task is normally terminated. In this case, messages are left in the queue regardless of the "CLR".

When "CHECK" is specified, since the messages other than the specified typed ones may be obtained, the type of

received message should be always checked.

Clear Message

tkse_clr_msg

C Language Interface

```
ER ercd = tkse_clr_msg (W t_mask, W last_mask);
```

Parameter

| | | |
|---|-----------|---|
| W | t_mask | message type mask targetd to clear (all message types in the case "MM_ALL" is specified) |
| W | last_mask | message type mask for clear exit (cleared to the end of the message queue in the case "MM_NULL" is specified) (only one message is cleared when "MM_ALL" is specified) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|-------|--|
| E_OK | normal completion |
| E_PAR | illegal parameter (t_mask<=0, last_mask<0) |

Description

Clears the received messages of the specified type to invoking process.

Out of the messages received in the message queue of invoking process, messages of the type specified by "t_mask" shall be cleared to the right before of the message of the type specified by "last_mask". The messages of the type specified by "last_mask" are not cleared. However, only one message of the "last_mask" type is cleared in the case "last_mask = MM_ALL".

Examples of specifying "t_mask" and the "last_mask" are shown as follows:

| t_mask | last_mask | behavior |
|--------|-----------|--|
| MM_ALL | MM_NULL | clear all the received messages |
| - | MM_ALL | clear only one message specified by "t_mask" |
| MM_ALL | MM_ALL | clear only the top one message |

Request Timeout Message

tkse_req_tmg

C Language Interface

```
ER ercd = tkse_req_tmg (TMO tmout, W code);
```

Parameter

| | | |
|-----|-------|---|
| TMO | tmout | time for sending message (milliseconds) |
| W | code | timeout message code |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|---------------------------------|
| E_OK | normal completion |
| E_PAR | illegal parameter (time <= 0) |
| E_SYSMEM | insufficient system memory area |

Description

Requests to send the timeout message (MS_TMOUT) to invoking process after specified time period is passed. This function is used to monitor the timeout of specific processings in combination with message handler.

Cancel Timeout Message

tkse_can_tmg

C Language Interface

```
ER ercd = tkse_can_tmg();
```

Parameter

none

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------|-------------------|
| E_OK | normal completion |
|------|-------------------|

Description

Cancels all the timeout message requests of invoking process. When there is no timeout message request, nothing shall be done.

The timeout messages that have been already sent and put in the message queue are not cleared.

Notify The Occurrence of Event

tkse_brk_msg

C Language Interface

```
ER ercd = tkse_brk_msg();
```

Parameter

none

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------|-------------------|
| E_OK | normal completion |
|------|-------------------|

Description

Releases the wait on “tkse_rcv_msg()” by the “WAIEVT” attribute specification.

In the case “tkse_brk_msg()” is invoked, if no task is put in the wait state by the “WAIEVT” specification, the request to release waiting is recorded. However, the request count to release waiting is not recorded.

Define Message Handler

tkse_def_msg

C Language Interface

```
ER ercd = tkse_def_msg (W t_mask, FUNCPC msg_hdr);
```

Parameter

| | | |
|--------|----------|--|
| W | t_mask | target message type mask |
| FUNCPC | msg_hdr | message handler start address |
| | NULL | release message handler definition |
| | MH_NONE | system definition handler (ignored) |
| | MH_BREAK | system definition handler (suspended) |
| | MH_TERM | system definition handler (process exit) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|--------|--|
| E_OK | normal completion |
| E_MACV | access to inaccessible address (msg_hdr) not allowed |
| E_PAR | illegal parameter (t_mask<=0) |

Description

Defines a message handler “ msg_hdr ” corresponding to the message of a specified type.

When a message handler for a message of the same type has already been defined, the message handler which was defined later becomes valid.

If msg_hdr = NULL, the defined message handler is released

Initial processes cannot use message handlers other than MH_NONE. When a message handler other than MH_NONE is specified, even if the message is received, the handler is not executed. Moreover, the specifications of MH_BREAK and MH_TERM are ignored.

Exit Message Handler

tkse_ret_msg

C Language Interface

```
ER ercd = tkse_ret_msg (W ret)
```

Parameter

| | | |
|---|-------|--|
| W | ret | return specification |
| | = 0: | resume the execution from the point interrupted by message handler |
| | != 0: | return from this system call to directly continue executing |

Return Parameter

| | | |
|----|------|------------------------------|
| ER | ercd | error code |
| | | (When "ret = 0") no return 0 |
| | | (When "ret != 0") =0 normal |

Error Code

| | |
|------|-------------------|
| E_OK | normal completion |
|------|-------------------|

Description

Terminates the execution of message handler.

When "ret = 0" is specified, the execution is resumed from the position interrupted by message handler instead of being returned from this system call. In this case, the execution does not return from tkse_ret_msg(). When an interruption occurs while executing a system call including a wait, an error code which indicates the invoking (start-up) of message handler is returned from the system call, in stead of ensuring the execution of the system call.

When "ret != 0" is specified, instead of being resumed at the position interrupted by the message handler, the task shall continue the execution after returning from this system. In this case, the control will usually be moved elsewhere by "longjmp()" at the end of the handler.

Regardless of the specification of ret, execution must be conducted at the end of processing of the message handler. Moreover, the execution must not be conducted in locations other than the message handler. Behavior when the execution is conducted in places other than the message handler is undefined.

If multiple message handler start requests occur, the requested message handler starts following the execution of tkse_ret_msg().

4.4 Global Name

4.4.1 Overview of the Global Name

The global name function of the “AMP TKSE” provides the functions to create data shared among processes by giving any name and to refer to the data by using the created name.

The global name data is a single 32-bit data (W), and the data can be named with up to 256 characters. Meaningful character codes are usually used as a name without any special restriction, and any data with up to 256 letters (512 bytes) until “TNULL(0)” can be used. Although meaningful names as TRON code are usually used for the names, it is possible to use data other than TRON code for the name if the termination is TNULL.

The data shared by global name can be referred to and changed from all processes. Moreover, change and deletion from processes other than the process which created global name can be prohibited.

The main use of global name is sharing of the following data, however, it is also possible to use global name for purposes other than the following.

- Process ID
- Address of the shared memory area
- Synchronization and communication object ID such as semaphore, message buffer and rendezvous
- Environment parameter used in the entire system

The global name function targets only the global name of its own AMP TKSE. It cannot operate the global names of other AMP TKSE.

The API specification of the global name function is the same as the T-Kernel TKSE Version 1.00 Specification

Although the sharing of data for the process and synchronization and communication object was described as the main use of the global name in T-Kernel TKSE Version 1.00 Specification, AMP TKSE has a process and object ID retrieval function, therefore, the use of AMP TKSE is recommended. However, global names can be used to share IDs for interchangeability with the existing software.

4.4.2 System Calls

Create Global Name Data

tkse_cre_nam

C Language Interface

```
ER ercd = tkse_cre_nam(TC* name, W data, W opt);
```

Parameter

| | | |
|-----|------|---|
| TC* | name | target global name (only top 256 characters (512 bytes) valid) |
| W | data | data to register |
| W | opt | specify data creation (N_CREATE N_MODIFY N_FORCE) [NA_PROTECT] [TA_DELEXIT] N_CREATE : create new N_MODIFY : modification N_FORCE : creation and modification NA_PROTECT : protect specification against modification and removal TA_DELEXIT : auto removal specification |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | normal completion |
| E_MACV | access to address (name) not allowed |
| E_OBJ | name already present (for N_CREATE) name is protected (for N_MODIFY, N_FORCE) |
| E_NOEXS | name does not exist (for N_MODIFY) |
| E_PAR | illegal parameter (illegal opt, blank name) |
| E_SYSMEM | insufficient system memory area |

Description

Creates or modifies global name data specified by the name.

"opt" specifies behavior of creation and change of global name.

If opt = N_CREATE is specified and the global name data with specified name is not present, then it is created. If the data is already present, an error occurs.

If opt = N_MODIFY is specified and the global name data with specified name is present, then the data is modified. If the data is not present, an error occurs.

If opt = N_FORCE is specified and the global name data with specified name is present, then the data is modified. If the data is already present, an error occurs.

NA_PROTECT as well as N_CREATE and N_Force may be specified. If NA_PROTECT is specified, processes other than those that created the global name data are prohibited from modifying and removing it. If this or tkse_del_nam() calls is issued from processes other than those that created the global name data to which NA_PROTECT is specified, E_OBJ error occurs. NA_PROTECT specification is valid until the object is removed.

TA_DELEXIT as well as other options can be specified. If TA_DELEXIT is specified, global name data will be removed automatically when the process which created or last modified the data exits. Even if TA_DELEXIT is already specified by other process, the last process which executed this system call by specifying TA_DELEXIT is processed.

Remove Global Name Data

tkse_del_nam

C Language Interface

```
ER ercd = tkse_del_nam(TC* name);
```

Parameter

| | | |
|-----|------|--|
| TC* | name | target global name (only top 256 characters (512 bytes) valid) |
|-----|------|--|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|--------------------------------------|
| E_OK | normal completion |
| E_MACV | access to address (name) not allowed |
| E_OBJ | name is protected |
| E_NOEXS | name does not exist |
| E_PAR | illegal parameter (blank name) |

Description

Removes global name data specified by the name.

If NA_PROTECT is specified during global name data creation and processes other than those that created the global name data try to remove it, E_OBJ error code is returned.

Get Global Name Data

tkse_get_nam

C Language Interface

```
ER ercd = tkse_get_nam (TC* name, W* data);
```

Parameter

| | | |
|-----|------|--|
| TC* | name | target global name (only up to the top 256 characters (512 bytes) valid) |
| W* | data | get data storage area |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | normal completion |
| E_MACV | access to inaccessible address (name, data) not allowed |
| E_NOEXS | name does not exist |
| E_PAR | illegal parameter (blank name) |

Description

Gets global name data specified by the name.

4.5 Synchronous Intertask Communication Function

4.5.1 Synchronous Intertask Communication Function Overview

The synchronous intertask communication function of the “AMP TKSE” provides the followings as a mechanism for synchronization and communication among tasks.

- semaphore
- mutex
- eventflag
- mailbox
- message buffer
- rendezvous

Each function is almost equal to the corresponding functions in AMP T-Kernel. However, they are not completely compatible since there are restrictions on parts of the functions.

Since the ID of each object ID such as semaphores is managed with AMP TKSE, it is different from IDs used by AMP T-Kernel system calls. For this reason, object IDs created with AMP TKSE cannot be used with AMP T-Kernel as they are. Moreover, objects created with AMP T-Kernel cannot be handled with AMP TKSE. However, task IDs are the same in AMP TKSE and AMP T-Kernel as an exception.

As for the task communication function, the functions of the object name and the access attribute are extended to the T-Kernel Standard Extension Version 1.00 Specification.

The object name is specified when the synchronization and communication object is created, and can be used to retrieve ID numbers.

Access from tasks to the synchronization and communication object is determined in accordance with the access attribute specified during creation. If access is not possible due to the access attribute, error code `E_DACV` will be returned (For details, refer to "2.3.3 Task Communication Function").

The task communication function can operate AMP TKSE objects of other kernels. In other words, task synchronization and communication between kernels can be conducted. However, the creation and deletion of objects only targets objects of its own AMP TKSE.

The following section describes only the differences (restrictions) with AMP T-Kernel system calls. Refer to AMP T-Kernel specifications for details of each system call.

4.5.2 System Calls (Semaphore)

Create Semaphore

tkse_cre_sem

C Language Interface

```
ID semid = tkse_cre_sem ( T_CSEM *pk_csem );
```

Parameter

T_CSEM *pk_csem semaphore creation information

```
typedef struct t_csem {
    VP      exinf;      /* extended information */
    ATR     sematr;     /* semaphore attribute */
    INT     isemcnt;    /* semaphore's initial count value */
    INT     maxsem;     /* semaphore's maximum count value */
    ID      domid;      /* domain ID (reserved) */
    UB      oname[8];   /* object name */
} T_CSEM;
```

semaphore attribute sematr

```
sematr: = (TA_TFIFO || TA_TPRI) | (TA_FIRST || TA_CNT) | TA_DELEXIT
          |( TA_GLOBAL  || TA_KLOCAL  || TA_PLOCAL) | TA_ONAME
```

| | |
|------------|--|
| TA_TFIFO | manage wait tasks with "FIFO" |
| TA_TPRI | manage wait tasks with priority order |
| TA_FIRST | prioritize a task at the top of the queue |
| TA_CNT | priotize tasks with few requests |
| TA_DELEXIT | specify auto deletion |
| TA_GLOBAL | specify global attribute to access attribute |
| TA_KLOCAL | specify Kernel Local to access attribute |
| TA_PLOCAL | specify Process Local to access attribute |
| TA_ONAME | specify the object name |

Return Parameter

| | | | |
|----|-------|-----|----------------------------------|
| ID | semid | > 0 | semaphore ID (normal completion) |
| | | < 0 | error code |

Error code

| | |
|---------|---|
| E_NOMEM | Insufficient Memory(Memory for control block cannot be allocated) |
| E_LIMIT | Number of semaphores exceeds the system limit |
| E_RSATR | Reserved attribute(sematr is invalid or cannot be used) |
| E_PAR | Parameter error(pk_csem is invalid, isemcnt or maxsem is negative or invalid) |
| E_ONAME | Specified object name has already been used |

Description

Creates a semaphore according to "pk_csem". However, "exinf" and "domid" are ignored.

In the case the "TA_DELEXIT" attribute is specified, the semaphore is automatically deleted when the process which created the semaphore exits. TA_GLOBAL, TA_KLOCAL, and the TA_PLOCAL attributes specify the access attributes of the semaphore. When the TA_PLOCAL attribute is specified, the TA_DELEXIT attribute is automatically specified as well.

Although the other attributes are equivalent to the AMP T-Kernel semaphore, the "TA_NODISWAI", "TA_DOMID", "TA_PUBLIC" "TA_PROTECTED" and "TA_PRIVATE" attributes cannot be specified.

Semaphores of other AMP TKSE cannot be created.

Delete Semaphore

tkse_del_sem

C Language Interface

```
ER ercd = tkse_del_sem( ID semid);
```

Parameter

| | | |
|----|-------|--------------|
| ID | semid | semaphore ID |
|----|-------|--------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (semid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the semaphore specified in semid does not exist) |
| E_DACV | Access protection violation |

Description

Deletes the semaphore specified by “semid”.

Semaphores of other AMP TKSE cannot be deleted.

Return Semaphore Resource

tkse_sig_sem

C Language Interface

```
ER ercd = tkse_sig_sem ( ID semid, INT cnt );
```

Parameter

| | | |
|-----|-------|----------------------------------|
| ID | semid | semaphore ID |
| INT | cnt | the number of returned resources |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (semid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the semaphore specified in semid does not exist) |
| E_QOVR | Queuing or nesting overflow (Overflow of queue count semcnt) |
| E_PAR | Parameter error (cnt <= 0) |
| E_DACV | Access protection violation |

Description

Returns as many resources as the number of "cnt" to semaphore specified by "semid".
Semaphores with the global attribute of other AMP TKSE can be specified.

Get Semaphore Resource

tkse_wai_sem

C Language Interface

```
ER ercd = tkse_wai_sem ( ID semid, INT cnt, TMO tmout );
```

Parameter

| | | |
|-----|-------|----------------------------------|
| ID | semid | semaphore ID |
| INT | cnt | the number of returned resources |
| TMO | tmout | timeout period |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (semid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the semaphore specified in semid does not exist) |
| E_PAR | Parameter error (tmout <= (-2), cnt < 0) |
| E_DLT | The object being waited for was deleted (the specified semaphore was deleted while waiting) |
| E_RLWAI | Wait state released |
| E_DISWAI | Wait released by wait disabled state |
| E_TMOUT | Polling failed or timeout |
| E_DACV | Access protection violation |

Description

Gets as many resources as the number of “cnt” from semaphore specified by “semid”.

When the task is interrupted by the message handler, the wait state is released and “E_DISWAI” is returned.

Semaphores with the global attribute of other AMP TKSE can be specified.

Refer to Semaphore State

tkse_ref_sem

C Language Interface

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

Parameter

T_RSEM *pk_rsem address to which the semaphore state is returned

```
typedef struct t_rsem {
    VP      exinf;      /* extended information */
    ID      wtsk;      /* waiting task ID */
    INT     semcnt;    /* current semaphore count value */
} T_RSEM;
```

Return Parameter

ER ercd error code

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (semid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the semaphore specified in semid does not exist) |
| E_PAR | Parameter error (address of the return parameter packet cannot be used) |
| E_DACV | Access protection violation |

Description

Refers to the semaphore state specified by “semid”, and return its content to the address indicated by “pk_rsem”. However, exinf always becomes NULL regardless of the specification when it was created. Semaphores with the global attribute of other AMP TKSE can be specified.

4.5.3 System Calls (Mutex)

Create Mutex

tkse_cre_mtx

C Language Interface

```
ID mtxid = tkse_cre_mtx ( T_CMTX *pk_cmtx );
```

Parameter

T_CMTX *pk_cmtx mutex creation information

```
typedef struct t_cmtx {
    VP      exinf;      /* extended information */
    ATR     mtxatr;     /* mutex attribute */
    PRI     ceilpri;    /* mutex's ceiling on priority level*/
    ID      domid;      /* domain ID(reserved) */
    UB      oname[8];   /* object name */
} T_CMTX;
```

mutex attribute mtxatr

```
mtxatr := (TA_TFIFO || TA_TPRI) | TA_DELEXIT
         |( TA_GLOBAL  || TA_KLOCAL  || TA_PLOCAL) | TA_ONAME
```

| | |
|------------|--|
| TA_TFIFO | manage wait tasks with "FIFO" |
| TA_TPRI | manage wait tasks with priority order |
| TA_DELEXIT | specify auto deletion |
| TA_GLOBAL | specify global attribute to access attribute |
| TA_KLOCAL | specify Kernel Local to access attribute |
| TA_PLOCAL | specify Process Local to access attribute |
| TA_ONAME | specify the object name |

Return Parameter

| | | | |
|----|-------|-----|------------------------------|
| ID | mtxid | > 0 | mutex ID (normal completion) |
| | | < 0 | error code |

Error Code

| | |
|---------|---|
| E_NOMEM | Insufficient memory(memory for control block cannot be allocated) |
| E_LIMIT | Number of mutex exceeded the system limit |
| E_RSATR | Reserved attribute (mtxatr is invalid or cannot be used) |
| E_PAR | Parameter error (pk_cmtx,ceilpri is invalid) |
| E_ONAME | Specified object name has already been used |

Description

Creates the mutex according to "pk_cmtx". However, "exinf", "ceilpri" and "domid" are ignored.

When the "TA_DELEXIT" attribute is specified, the mutex is automatically deleted when the process which created the mutex exits. The TA_GLOBAL, TA_KLOCAL, and TA_PLOCAL attribute specify the access attribute of the mutex. When the TA_PLOCAL attribute is specified, the TA_DELEXIT attribute is automatically specified as well.

Although other attributes are equivalent to the AMP T-Kernel mutex, the "TA_NODISWAI", "TA_INHERIT", "TA_CEILING", "TA_DOMID", "TA_PUBLIC", "TA_PROTECTED" and "TA_PRIVATE" attributes cannot be specified.

Mutexes of other AMP TKSE cannot be created.

Delete Mutex**tkse_del_mtx****C Language Interface**

```
ER ercd = tkse_del_mtx (ID mtxid)
```

Parameter

| | | |
|----|-------|----------|
| ID | mtxid | mutex ID |
|----|-------|----------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mtxid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the mutex specified in mtxid does not exist) |
| E_DACV | Access protection violation |

Description

Deletes the mutex specified by "mtxid".

Mutexes of other AMP TKSE cannot be deleted.

Lock Mutex

tkse_loc_mtx

C Language Interface

```
ER ercd = tkse_loc_mtx ( ID mtxid, TMO tmout );
```

Parameter

| | | |
|-----|-------|----------------|
| ID | mtxid | mutex ID |
| TMO | tmout | timeout period |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mtxid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the mutex specified in mtxid does not exist) |
| E_PAR | Parameter error (tmout <= (-2)) |
| E_DLT | The object being waited for was deleted (the mutex was deleted while waiting for a lock) |
| E_RLWAI | Wait state released |
| E_DISWAI | Wait released by wait disabled state |
| E_TMOUT | Polling failed or timeout |
| E_ILUSE | Illegal use (multiple lock, or upper priority limit exceeded) |
| E_DACV | Access protection violation |

Description

Locks the mutex specified by "mtxid".

When the task is interrupted by the message handler, the wait state is released and "E_DISWAI" is returned.

Mutexes with the global attribute of other AMP TKSE can be specified.

Unlock Mutex

tkse_unl_mtx

C Language Interface

```
ER ercd = tkse_unl_mtx ( ID mtxid );
```

Parameter

| | | |
|----|-------|----------|
| ID | mtxid | mutex ID |
|----|-------|----------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mtxid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the mutex specified in mtxid does not exist) |
| E_ILUSE | Illegal use (not a mutex locked by the invoking task) |
| E_DACV | Access protection violation |

Description

Unlocks the mutex specified by "mtxid".

Mutexes with the global attribute of other AMP TKSE can be specified.

Refer to Mutex State

tkse_ref_mtx

C Language Interface

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

Parameter

| | | |
|--------|----------|--|
| ID | mtxid | mutex ID |
| T_RMTX | *pk_rmtx | the address to which the mutex state is returned |

```
typedef struct t_rmtx {
    VP      exinf;      /* extended information */
    ID      htsk;      /* locked task ID */
    ID      wtsk;      /* lock wait task ID */
} T_RMTX;
```

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mtxid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the mutex specified in mtxid does not exist) |
| E_PAR | Parameter error (the address of the return parameter packet cannot be used) |
| E_DACV | Access protection violation |

Description

Refers to the mutex state specified by “mtxid”, and returns its content to the address indicated by “pk_rmtx”. However, NULL is always returned to “exinf”.

Mutexes with the global attribute of other AMP TKSE can be specified.

4.5.4 System Calls (Eventflag)

Create Eventflag

tkse_cre_flg

C Language Interface

```
ID flgid = tkse_cre_flg ( T_CFLG *pk_cflg )
```

Parameter

T_CFLG *pk_cflg eventflag create information

```
typedef struct t_cflg {
    VP      exinf;          /* extended information */
    ATR     flgatr;        /* eventflag attribute */
    UINT    iflgptn;       /* eventflag initial value */
    ID      domid;         /* domain ID (reserved) */
    UB      oname[8];      /* object name */
} T_CFLG;
```

eventflag attribute flgatr

```
flgatr: = (TA_TFIFO || TA_TPRI) | (TA_WMUL || TA_WSGL) | TA_DELEXIT
          |( TA_GLOBAL  || TA_KLOCAL  || TA_PLOCAL) | TA_ONAME
```

| | |
|------------|--|
| TA_TFIFO | manage wait tasks with "FIFO" |
| TA_TPRI | manage wait tasks with priority order |
| TA_WSGL | disallow a wait on multiple tasks |
| TA_WMUL | allow a wait on multiple tasks |
| TA_DELEXIT | specify auto deletion |
| TA_GLOBAL | specify global attribute to access attribute |
| TA_KLOCAL | specify Kernel Local to access attribute |
| TA_PLOCAL | specify Process Local to access attribute |
| TA_ONAME | specify the object name |

Return Parameter

| | | | |
|----|-------|-----|----------------------------------|
| ID | flgid | > 0 | eventflag ID (normal completion) |
| | | < 0 | error code |

Error Code

| | |
|---------|--|
| E_NOMEM | Insufficient memory (memory for control block cannot be allocated) |
| E_LIMIT | Number of event flags exceeded system limit |
| E_RSATR | Reserved attribute (flgatr is invalid or cannot be used) |
| E_PAR | Parameter error(pk_cfg is invalid) |
| E_ONAME | Specified object name has already been used. |

Description

Creates an eventflag according to "pk_cfg". However, "exinf" and "domid" are ignored.

flgatr specifies the attribute of the event flag to be created. In the case the "TA_DELEXIT" attribute is specified, the eventflag is automatically deleted when the process which created the eventflag exits. The TA_GLOBAL, TA_KLOCAL, and TA_PLOCAL attribute specify the access attribute of the event flag. When the TA_PLOCAL attribute is specified, the TA_DELEXIT attribute is automatically specified.

Although the other attributes are equivalent to the AMP T-Kernel eventflag, the "TA_NODISWAI" "TA_DOMID", "TA_PUBLIC", "TA_PROTECTED" and the "TA_PRIVATE" attributes cannot be specified.

Event flags of other AMP TKSE cannot be created.

Delete Eventflag

tkse_del_flg

C Language Interface

```
ER ercd = tkse_del_flg ( ID flgid );
```

Parameter

| | | |
|----|-------|--------------|
| ID | flgid | eventflag ID |
|----|-------|--------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|--|
| E_OK | Normal completion |
| E_ID | Invalid ID number (flgid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (domain of domid does not exist) |
| E_DACV | Access protection violation |

Description

Deletes the eventflag specified by “flgid”.

Event flags of other AMP TKSE cannot be deleted.

Set Eventflag

tkse_set_flg

C Language Interface

```
ER ercd = tkse_set_flg ( ID flgid, UINT setptn );
```

Parameter

| | | |
|------|--------|-----------------------|
| ID | flgid | eventflag ID |
| UINT | setptn | bit pattern to be set |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|--|
| E_OK | Normal completion |
| E_ID | Invalid ID number (flgid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the event flag specified in flgid does not exist) |
| E_DACV | Access protection violation |

Description

Sets the pattern of “setptn” to the eventflag specified by “flgid”.

Event flags with the global attribute of other AMP TKSE can be specified.

Clear Eventflag

tkse_clr_flg

C Language Interface

```
ER ercd = tkse_clr_flg ( ID flgid, UINT clrptn );
```

Parameter

| | | |
|------|--------|---------------------------|
| ID | flgid | eventflag ID |
| UINT | clrptn | bit pattern to be cleared |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|--|
| E_OK | Normal completion |
| E_ID | Invalid ID number (flgid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the event flag specified in flgid does not exist) |
| E_DACV | Access protection violation |

Description

The eventflag specified by “flgid” shall be cleared with the pattern of “clrptn”.
Event flags with the global attribute of other AMP TKSE can be specified.

Wait Eventflag

tkse_wai_flg

C Language Interface

```
ER ercd = tkse_wai_flg ( ID flgid, UINT waiptn, UINT wfmode, UINT *p_flgptn, TMO tmout );
```

Parameter

| | | |
|------|------------|---|
| ID | flgid | eventflag ID |
| UINT | waiptn | wait bit pattern |
| UINT | wfmode | wait mode |
| | TWF_ANDW | AND wait |
| | TWF_ORW | OR wait |
| | TWF_CLR | specify to clear all |
| | TWF_BITCLR | specify to clear conditional bit only |
| UINT | *p_flgptn | the address to which bit pattern is returned when a wait is cleared |
| TMO | tmout | timeout period |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | Normal completion |
| E_ID | Invalid ID number (flgid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the event flag specified in flgid does not exist) |
| E_PAR | Parameter error (waiptn = 0, wfmode is invalid, or tmout <= (-2)) |
| E_OBJJ | Invalid object state (multiple tasks are waiting for an event flag with TA_WSGL attribute) |
| E_DLT | The object being waited for was deleted (the specified event flag was deleted while waiting) |
| E_RLWAI | Wait state released |
| E_DISWAI | Wait released by wait disabled state |
| E_TMOUT | Polling failed or timeout |
| E_DACV | Access protection violation |

Description

Waits until the bit specified by “waiptn” is set to the eventflag specified by “flgid” in the wait condition specified by “wfmode”.

When the task is interrupted by the message handler, the wait state is released and “E_DISWAI” is returned.

Event flags with the global attribute of other AMP TKSE can be specified.

Refer to Eventflag State

tkse_ref_flg

C Language Interface

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

Parameter

T_RFLG *pk_rflg the address to which the eventflag state is returned

```
typedef struct t_rflg {
    VP      exinf;      /* extended information */
    ID      wtsk;      /* waiting task ID */
    UINT    flgptn;    /* current eventflag pattern */
} T_RFLG;
```

Return Parameter

ER ercd error code

Error Codes

| | |
|---------|--|
| E_OK | Normal completion |
| E_ID | Invalid ID number (flgid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the event flag specified in flgid does not exist) |
| E_PAR | Parameter error (pk_msg is a value that cannot be used) |
| E_DACV | Access protection violation |

Description

Refers to the semaphore state specified by “flgid” and return its content to the address indicated by “pk_rflg”. However, NULL is always returned to “exinf”.

Event flags with the global attribute of other AMP TKSE can be specified.

4.5.5 System Calls (Mailbox)

Create Mailbox

tkse_cre_mbx

C Language Interface

```
ID mbxid = tkse_cre_mbx( T_CMBX *pk_cmbx );
```

Parameter

T_CMBX* pk_cmbx mailbox creation information

```
typedef struct t_cmbx {
    VP      exinf;          /* extended information */
    ATR     mbxatr;        /* mailbox attribute */
    ID      domid;         /* domain ID (reserved) */
    UB      oname[8];      /* object name */
} T_CMBX;
```

mailbox attribute mbxatr

```
mbxatr: = (TA_TFIFO || TA_TPRI) | (TA_MFIFO || TA_MPRI) | TA_ONAME
```

| | |
|----------|--|
| TA_TFIFO | queueing of waiting tasks is in FIFO |
| TA_TPRI | queueing of waiting tasks is in priority order |
| TA_MFIFO | queueing of messages is in FIFO |
| TA_MPRI | queueing of messages is in priority order |
| TA_ONAME | specify the object name |

Return Parameter

| | | | |
|----|-------|-----|--------------------------------|
| ID | mbxid | > 0 | mailbox ID (normal completion) |
| | | < 0 | error code |

Error Code

| | |
|---------|--|
| E_NOMEM | Insufficient memory (memory for a control block or buffer cannot be allocated) |
| E_LIMIT | Maximum number of mailboxes of the system exceeded |
| E_RSATR | Reserved attribute (mbxatr is invalid or cannot be used) |
| E_PAR | Parameter error (pk_cmbx is invalid) |
| E_ONAME | Specified object name has already been used |

Description

Creates the mailbox according to `pk_cmbx`. However, `exinf` and `domid` are ignored.

The attribute specification is equivalent to AMP T-Kernel mailbox, but `TA_NODISWAI`, `TA_DOMID`, `TA_PUBLIC`, `TA_PROTECTED` and `TA_PRIVATE` attributes cannot be specified. Also, if `TA_DELEXIT` is specified, it will be ignored.

Since the pointer to the data is passed, the mailbox is only available within the process which created the mailbox. It is not available in communications between different processes. Therefore, all mailboxes are process local attributes, and when the created process ends, the mailboxes are automatically deleted.

Remove Mailbox

tkse_del_mbx

C Language Interface

```
ER ercd = tkse_del_mbx ( ID mbxid );
```

Parameter

| | | |
|----|-------|------------|
| ID | mbxid | mailbox ID |
|----|-------|------------|

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mbxid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the mailbox specified in mbxid does not exist) |
| E_DACV | Access protection violation |

Description

Removes the mailbox denoted by mbxid.

If processes other than the process which created the mailbox try to remove the mailbox, an access protection error occurs.

Send to Mailbox

tkse_snd_mbx

C Language Interface

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

Parameter

| | | |
|--------|--------|---------------------------------|
| ID | mbxid | mailbox ID |
| T_MSG* | pk_msg | start address of message packet |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mbxid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the mailbox specified in mbxid does not exist) |
| E_PAR | Parameter error (pk_msg is a value that cannot be used) |
| E_DACV | Access protection violation |

Description

Sends the message packet whose start address is pk_msg to the target mailbox denoted by mbxid.

The content of message packet is not copied and only start address (pk_msg value) is passed on receiving.

If processes other than the process which created the mailbox try to send to the mailbox, an access protection error occurs.

Receive from Mailbox

tkse_rcv_mbx

C Language Interface

```
ER ercd = tkse_rcv_mbx ( ID mbxid, T_MSG **ppk_msg, TMO tmout ) ;
```

Parameter

| | | |
|-----|-------|-----------------------|
| ID | mbxid | mailbox ID |
| TMO | tmout | timeout specification |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

| | | |
|--------|--------|---------------------------------|
| T_MSG* | pk_msg | start address of message packet |
|--------|--------|---------------------------------|

Error Code

| | |
|----------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mbxid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the mailbox specified in mbxid does not exist) |
| E_PAR | Parameter error (tmout <= (-2)) |
| E_DLT | The object being waited for was deleted (the mailbox was deleted while waiting) |
| E_RLWAI | Wait state released |
| E_DISWAI | Wait released by wait disabled state |
| E_TMOUT | Polling failed or timeout |
| E_DACV | Access protection violation |

Description

Receives a message from the mailbox denoted by mbxid.

If processes other than the process which created the mailbox try to receive from the mailbox, an access protection error occurs.

Refer to Mailbox State

tkse_ref_mbx

C Language Interface

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

Parameter

| | |
|-----------------|---|
| ID mbxid | mailbox ID |
| T_RMBX* pk_rmbx | packet address to which the mailbox state is returned |

```
typedef struct t_rmbx {
    VP exinf;          /* extended information */
    ID wtsk;          /* presence of waiting task */
    T_MSG* pk_msg;    /* start address of the next message packet to be received */
} T_RMBX;
```

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mbxid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the mailbox specified in mbxid does not exist) |
| E_PAR | Parameter error (the return parameter packet address cannot be used) |
| E_DACV | Access protection violation |

Description

Refers to the mailbox state specified by mbxid and returns its content to the address denoted by pk_rmbx . However, exinf always becomes NULL regardless of the specification when it was created.

4.5.6 System Calls (Message buffer)

Create Message buffer

tkse_cre_mbf

C Language Interface

```
ID mbfid = tkse_cre_mbf ( T_CMBF *pk_cmbf );
```

Parameter

```
T_CMBF *pk_cmbf      message buffer create information
typedef struct t_cmbf {
    VP      exinf;      /* extended information */
    ATR      mbfatr;    /* message buffer attribute */
    INT      bufsz;     /* message buffer size (bytes) */
    INT      maxmsz;    /* maximum length of message (bytes) */
    ID       domid;     /* domain ID (reserved) */
    UB       oname[8];  /* object name */
} T_CMBF;

message buffer attribute mbfatr
mbfatr := (TA_TFIFO || TA_TPRI) | TA_DELEXIT
          |(TA_GLOBAL  || TA_KLOCAL  || TA_PLOCAL) | TA_ONAME
          TA_TFIFO    manage wait tasks with "FIFO"
          TA_TPRI     manage wait tasks with priority order
          TA_DELEXIT  specify auto deletion
          TA_GLOBAL   specify global attribute to access attribute
          TA_KLOCAL   specify Kernel Local to access attribute
          TA_PLOCAL   specify Process Local to access attribute
          TA_ONAME    specify the object name
```

Return Parameter

```
ID      mbfid      > 0      message buffer ID (normal completion)
                   < 0      error code
```

Error Code

```
E_NOMEM    Insufficient memory (memory for a control block or ring buffer cannot be allocated)
E_LIMIT     Maximum number of message buffers of the system exceeded
```


| | |
|---------|--|
| E_RSATR | Reserved attribute (mbfatr is invalid or cannot be used) |
| E_PAR | Parameter error (pk_cmbf is invalid, bufsz or maxmsz is negative or invalid) |
| E_ONAME | Specified object name has already been used |

Description

Creates message buffer according to "pk_cmbf". However, "exinf" and "domid" ignored.

mbfatr specifies the message buffer attribute. When the "TA_DELEXIT" attribute is specified, the message buffer is automatically deleted when the process which created the message buffer exits. The TA_GLOBAL, TA_KLOCAL, and TA_PLOCAL attribute specify the access attribute of the message buffer. When the TA_PLOCAL attribute is specified, the TA_DELEXIT attribute is automatically specified.

Although the other attributes are equivalent to the AMP T-Kernel message buffer, the "TA_NODISWAI", "TA_DOMID", "TA_PUBLIC", "TA_PROTECTED" and "TA_PRIVATE" attributes cannot be specified.

Message buffers of other AMP TKSE cannot be created.

Delete Message buffer

tkse_del_mbf

C Language Interface

```
ER ercd = tkse_del_mbf ( ID mbfid );
```

Parameter

| | | |
|----|-------|-------------------|
| ID | mbfid | message buffer ID |
|----|-------|-------------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mtxid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the mutex specified in mtxid does not exist) |
| E_DACV | Access protection violation |

Description

Deletes the message buffer specified by "mbfid".

Message buffers of other AMP TKSE cannot be deleted.

Send to Message buffer

tkse_snd_mbf

C Language Interface

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

Parameter

| | | |
|-----|-------|---------------------------------|
| ID | mbfid | message buffer |
| VP | msg | start address of a sending data |
| INT | msgsz | size of a sending data (bytes) |
| TMO | tmout | timeout period |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mbfid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the message buffer specified in mbfid does not exist) |
| E_PAR | Parameter error (msgsz < 0, msgsz > maxmsz, value in msg can not be used, or tmout <= (-2)) |
| E_DLT | The object being waited for was deleted (message buffer was deleted while waiting) |
| E_RLWAI | Wait state released (tk_rel_wai received in wait state) |
| E_DISWAI | Wait released by wait disabled state |
| E_TMOUT | Polling failed or timeout |
| E_DACV | Access protection violation |

Description

Sends data specified by “msg” and “msgsz” to the message buffer specified by “mbfid”.

When the task is interrupted by the message handler, the wait state is released and “E_DISWAI” is returned.

Message buffers with the global attribute of other AMP TKSE can be specified.

Receive from Message buffer

tkse_rcv_mbf

C Language Interface

```
INT msgsz = tkse_rcv_mbf ( ID mbfid, VP msg, TMO tmout );
```

Parameter

| | | |
|-----|-------|--------------------------------------|
| ID | mbfid | message buffer |
| VP | msg | start address to store received data |
| TMO | tmout | timeout period |

Return Parameter

| | | | |
|-----|-------|-----|--------------------------------|
| INT | msgsz | > 0 | size of receiving data (bytes) |
| | | < 0 | error code |

Error Code

| | |
|----------|--|
| E_ID | Invalid ID number (mbfid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the message buffer specified in mbfid does not exist) |
| E_PAR | Parameter error (value in msg cannot be used, or tmout <= (-2)) |
| E_DLT | The object being waited for was deleted (message buffer was deleted while waiting) |
| E_RLWAI | Wait state released (tk_rel_wai received in wait state) |
| E_DISWAI | Wait released by wait disabled state |
| E_TMOUT | Polling failed or timeout |
| E_DACV | Access protection violation |

Description

Receives data from the message buffer specified by “mbfid” to store in “msg”.

When the task is interrupted by the message handler, the wait state is released and “E_DISWAI” is returned.

Message buffers with the global attribute of other AMP TKSE can be specified.

Refer to Message Buffer State

tkse_ref_mbf

C Language Interface

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

Parameter

| | | |
|--------|----------|----------------------------------|
| ID | mbfid | message buffer |
| T_RMBF | *pk_rmbf | message buffer state information |

```
typedef struct t_rmbf {
    VP      exinf;      /* extended information */
    ID      wtsk;      /* receive wait task ID of the task waiting to be received*/
    ID      stsk;      /* send wait task ID of the task waiting to be sent*/
    INT     msgsz;     /* size of the next message to be received (bytes) */
    INT     frbufsz;   /* free buffer size (bytes) */
    INT     maxmsz;    /* maximum length of message (bytes) */
} T_RMBF;
```

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|--|
| E_OK | Normal completion |
| E_ID | Invalid ID number (mbfid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the message buffer specified in mbfid does not exist) |
| E_PAR | Parameter error (the address of the return parameter packet cannot be used) |
| E_DACV | Access protection violation |

Description

Refers to the message buffer state specified by “mbfid” and return its content to the address indicated by “pk_rmbf”.

However, NULL is always returned to “exinf”.

Message buffers with the global attribute of other AMP TKSE can be specified.

4.5.7 System Calls (Rendezvous Port)

Create Rendezvous Port

tkse_cre_por

C Language Interface

```
ID porid = tkse_cre_por ( T_CPOR *pk_cpor );
```

Parameter

T_CPOR *pk_cpor rendezvous port create information

```
typedef struct t_cpor {
    VP      exinf;      /* extended information */
    ATR     poratr;     /* port attribute */
    INT     maxcmsz;    /* maximum length of call out message (bytes) */
    INT     maxrmsz;    /* maximum length of response message (bytes) */
    ID      domid;      /* domain ID(reserved) */
    UB      oname[8];   /* object name */
} T_CPOR;
```

rendezvous port attribute poratr

```
poratr = (TA_TFIFO || TA_TPRI) | TA_DELEXIT
          |( TA_GLOBAL  || TA_KLOCAL  || TA_PLOCAL) | TA_ONAME
```

| | |
|------------|--|
| TA_TFIFO | manage wait tasks with FIFO |
| TA_TPRI | manage wait tasks with priority order |
| TA_DELEXIT | auto removal specification |
| TA_GLOBAL | specify global attribute to access attribute |
| TA_KLOCAL | specify Kernel Local to access attribute |
| TA_PLOCAL | specify Process Local to access attribute |
| TA_ONAME | specify the object name |

Return Parameter

| | | | |
|----|-------|-----|-----------------------------------|
| ID | porid | > 0 | rendezvous ID (normal completion) |
| | | < 0 | error code |

Error Code

| | |
|---------|---|
| E_NOMEM | Insufficient memory (memory for control block cannot be allocated) |
| E_LIMIT | Maximum number of rendezvous ports of the system exceeded |
| E_RSATR | Reserved attribute (poratr is invalid or cannot be used) |
| E_PAR | Parameter error (pk_cpor is invalid, maxcmsz or maxrmsz is negative or invalid) |
| E_ONAME | Specified object has already been used |

Description

Creates rendezvous port according to "pk_cpor". However, "exinf" and "domid" is ignored.

poratr specifies the attribute of the rendezvous port. When the "TA_DELEXIT" attribute is specified, the rendezvous port is automatically deleted when the process which created the rendezvous port exits. TA_GLOBAL, TA_KLOCAL, and the TA_PLOCAL attribute specify the access attributes of the rendezvous port. When the TA_PLOCAL attribute is specified, the TA_DELEXIT attribute is automatically specified as well.

Although the other attributes are equivalent to the AMP T-Kernel eventflag, the "TA_NODISWAI", "TA_DOMID", "TA_PUBLIC", "TA_PROTECTED" and "TA_PRIVATE" attributes cannot be specified.

Rendezvous ports of other AMP TKSE cannot be created.

Delete Rendezvous Port

tkse_del_por

C Language Interface

```
ER ercd = tkse_del_por ( ID porid );
```

Parameter

| | | |
|----|-------|--------------------|
| ID | porid | rendezvous port ID |
|----|-------|--------------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (porid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the rendezvous port specified in porid does not exist) |
| E_DACV | Access protection violation |

Description

Deletes the rendezvous port specified by “porid”
 Rendezvous ports of other AMP TKSE cannot be deleted.

Call Rendezvous

tkse_cal_por

C Language Interface

```
INT rmsgsz = tkse_cal_por ( ID porid, UINT calptn, VP msg, INT cmsgsz, TMO tmout );
```

Parameter

| | | |
|------|--------|--|
| ID | porid | rendezvous port ID |
| UINT | calptn | bit pattern to designate selection condition |
| VP | msg | start address of message |
| INT | cmsgsz | size of calling message (bytes) |
| TMO | tmout | timeout period |

Return Parameter

| | | | |
|-----|--------|-----|--|
| INT | rmsgsz | > 0 | size of response message (the number of bytes) |
| | | < 0 | error code |

Error Code

| | |
|----------|--|
| E_ID | Invalid ID number (porid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the rendezvous port specified in porid does not exist) |
| E_PAR | Parameter error (cmsgsz < 0, cmsgsz > maxcmsz, calptn = 0, value that cannot be used in msg, tmout <= (-2)) |
| E_DLT | The object being waited for was deleted (the rendezvous port was deleted while waiting) |
| E_RLWAI | Wait state released (tk_rel_wai received in wait state) |
| E_DISWAI | Wait released by wait disabled state |
| E_TMOUT | Polling failed or timeout |
| E_DACV | Access protection violation |

Description

Calls the rendezvous for the rendezvous port specified by "porid".

When the task is interrupted by the message handler, the wait state is released and "E_DISWAI" is returned.

Rendezvous ports with the global attribute of other AMP TKSE can be specified.

Accept Rendezvous

tkse_acp_por

C Language Interface

```
INT msgsz = tkse_acp_por ( ID porid, UINT acpbtn, RNO *p_rdvno, VP msg, TMO tmout );
```

Parameter

| | | |
|------|----------|--|
| ID | porid | rendezvous port ID |
| UINT | acpbtn | bit pattern to designate selection condition |
| INT | *p_rdvno | the address to which the rendezvous number is returned |
| VP | msg | start address of message |
| TMO | tmout | timeout period |

Return Parameter

| | | | |
|-----|-------|-----|---------------------------------|
| INT | msgsz | > 0 | size of calling message (bytes) |
| | | < 0 | error code |

Error Code

| | |
|----------|---|
| E_ID | Invalid ID number (porid is invalid or cannot be used, or porid is a rendezvous port of another node) |
| E_NOEXS | Object does not exist (the rendezvous port specified in porid does not exist) |
| E_PAR | Parameter error (acpbtn = 0, value that cannot be used in msg, or tmout <=(-2)) |
| E_DLT | The object being waited for was deleted (the rendezvous port was deleted while waiting) |
| E_RLWAI | Wait state released (tk_rel_wai received in wait state) |
| E_DISWAI | Wait released by wait disabled state |
| E_TMOUT | Polling failed or timeout |
| E_DACV | Access protection violation |

Description

Accepts the rendezvous for the rendezvous port specified by "porid".

When the task is interrupted by the message handler, the wait state is released and "E_DISWAI" is returned.

Rendezvous ports with the global attribute of other AMP TKSE can be specified.

Forward Rendezvous

tkse_fwd_por

C Language Interface

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

Parameter

| | | |
|------|--------|--|
| ID | porid | rendezvous port ID |
| UINT | calptn | bit pattern to designate selection condition |
| INT | rdvno | rendezvous number before forwarding |
| VP | msg | start address of message |
| INT | cmsgsz | calling message size (bytes) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (porid is invalid or cannot be used, or porid is a rendezvous port of another node) |
| E_NOEXS | Object does not exist (the rendezvous port specified in porid does not exist) |
| E_PAR | Parameter error (cmsgsz < 0, cmsgsz > maxcmsz after forwarding, cmsgsz > maxrmsz before forwarding, calptn = 0, or msg has a value that cannot be used) |
| E_OBJ | Invalid object state (rdvno is invalid, or maxrmsz (after forwarding) > maxrmsz (before forwarding)) |
| E_DISWAI | Wait released by wait disabled state |
| E_DACV | Access protection violation |

Description

Forwards the accepted rendezvous to another rendezvous port.

Rendezvous ports with the global attribute of other AMP TKSE can be specified.

Reply to Rendezvous

tkse_rpl_rdv

C Language Interface

```
ER ercd = tkse_rpl_rdv ( RNO rdvno, VP msg, INT rmsgsz );
```

Parameter

| | | |
|-----|--------|--|
| INT | rdvno | rendezvous number |
| VP | msg | start address of response reply message |
| INT | rmsgsz | size of reply message size (the number of bytes) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|-------|--|
| E_OK | Normal completion |
| E_PAR | Parameter error (rmsgsz < 0, rmsgsz > maxrmsgsz, or value in msg cannot be used) |
| E_OBJ | Invalid object state (rdvno is invalid) |

Description

Returns rendezvous response reply to exit rendezvous.

Refer to Rendezvous Port State

tkse_ref_por

C Language Interface

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

Parameter

| | | |
|--------|----------|-----------------------------------|
| ID | porid | rendezvous port ID |
| T_RPOR | *pk_rpor | rendezvous port state information |

```
typedef struct t_rpor {
    VP      exinf;      /* extended information */
    ID      wtsk;      /* call wait task ID */
    ID      atsk;      /* accept wait task ID */
    INT     maxcmsz;   /* maximum length of call out message (bytes) */
    INT     maxrmsz;   /* maximum length of response message (bytes) */
} T_RPOR;
```

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_ID | Invalid ID number (porid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the rendezvous port specified in porid does not exist) |
| E_PAR | Parameter error (the return parameter packet address cannot be used) |
| E_DACV | Access protection violation |

Description

Refers to the message buffer state specified by “porid” and returns its content to the address indicated by “pk_rpor”.

However, NULL is always returned to “exinf”.

Rendezvous ports with the global attribute of other AMP TKSE can be specified.

4.6 Object Management

4.6.1 Overview

The object management function of AMP TKSE provides a kernel domain ID reference and ID number retrieval function.

The kernel domain ID is an identification number for identifying the kernel. It is also used to identify AMP TKSE. The kernel domain ID is automatically allocated by the kernel. The kernel domain ID allocated to the processor can be acquired by specifying the processor number.

The ID number retrieval function retrieves ID numbers by using the object name of the process and the synchronization and communication object. Only objects that can be operated from itself can be retrieved. In other words, the following objects can be retrieved.

- All objects with global attributes
- For objects with the kernel local attribute, objects of the same AMP TKSE as itself
- For objects with the process local attribute, objects of the same process as itself

The ID number retrieval specifies the object name and the target of retrieval.

When objects with the global attribute or kernel local attribute are retrieved, the kernel domain ID of AMP TKSE that the object belongs to is specified in the target of the retrieval. When objects with the process local attribute are retrieved, only its own process can be specified.

Although the access attribute cannot be specified for processes during creation, processes are treated as global attributes in the object management function.

4.6.2 System Calls

Get kernel domain ID

tkse_get_kdm

C Language Interface

```
ID domid = tkse_get_kdm( ID prcid );
```

Parameter

| | | |
|----|-------|---------------------|
| ID | prcid | Processor ID number |
|----|-------|---------------------|

Return Parameter

| | | |
|----|-----------|--------------------------------------|
| ID | domid > 0 | Kernel domain ID (normal completion) |
| | < 0 | error code |

Error Code

| | |
|------|--|
| E_ID | Invalid ID number (prcid is invalid or cannot be used) |
|------|--|

Description

Acquires the kernel domain ID number of the AMP T-Kernel that operates by the processor shown by prcid.

If prcid = PRC_SELF = 0, its own kernel domain ID will be returned.

The Kernel domain ID can be used to identify AMP T-Kernel and AMP TKSE operating on the AMP T-Kernel.

Object ID Retrieval

**tkse_fnd_prc, tkse_fnd_sem, tkse_fnd_flg, tkse_fnd_mbx,
tkse_fnd_mbf, tkse_fnd_mtx, tkse_fnd_por**

C Language Interface

```
ID prcid = tkse_fnd_prc ( ID domid, UB *oname ) ; /* process */
ID semid = tkse_fnd_sem ( ID domid, UB *oname ) ; /* semafor */
ID flgid = tkse_fnd_flg ( ID domid, UB *oname ) ; /* event flag */
ID mbxid = tkse_fnd_mbx ( ID domid, UB *oname ) ; /* mailbox */
ID mbfid = tkse_fnd_mbf ( ID domid, UB *oname ) ; /* message buffer */
ID mtxid = tkse_fnd_mtx ( ID domid, UB *oname ) ; /* mutex */
ID porid = tkse_fnd_por ( ID domid, UB *oname ) ; /* rendezvous port */
```

Parameter

| | | |
|-----|-------|------------------|
| ID | domid | Kernel domain ID |
| UB* | oname | Object name |

Return Parameter

| | | | |
|----|------|-----|---|
| ID | ~ id | > 0 | Specified object ID (normal completion) |
| | | < 0 | Error code |

Error Code

| | |
|---------|--|
| E_ID | Invalid ID number (domid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (object of oname does not exist) |
| E_PAR | Parameter error (oname is invalid or cannot be used) |

Description

Retrieves the object that belongs to the kernel or the process shown by domid by the object name, and gets the object ID. Specifies the object name of the object to be retrieved in oname.

In order to retrieve the object of other AMP TKSE, specify the kernel domain ID of the AMP TKSE in domid. Objects with the global attribute of a specified AMP TKSE become the target of the retrieval.

If domid=TAR_SELF(=0) is specified, objects with the global attribute and kernel local attribute in its own AMP TKSE become the target of the retrieval.

If domid=TAR_PRIV(=-1) is specified, objects with the process local attribute of its own process become the target.

If the domain ID of its own AMP TKSE is specified, it is the same as when TAR_SELF is specified.

If the object specified in domid and oname is found, the ID of the object is returned. If the corresponding object does not exist, E_NOEXS is returned.

Object ID Retrieval

**tkse_fnd_prc, tkse_fnd_sem, tkse_fnd_flg, tkse_fnd_mbx,
tkse_fnd_mbf, tkse_fnd_mtx, tkse_fnd_por**

C Language Interface

```
ID prcid = tkse_fnd_prc ( ID domid, UB *oname ) ; /* process */
ID semid = tkse_fnd_sem ( ID domid, UB *oname ) ; /* semafor */
ID flgid = tkse_fnd_flg ( ID domid, UB *oname ) ; /* event flag */
ID mbxid = tkse_fnd_mbx ( ID domid, UB *oname ) ; /* mailbox */
ID mbfid = tkse_fnd_mbf ( ID domid, UB *oname ) ; /* message buffer */
ID mtxid = tkse_fnd_mtx ( ID domid, UB *oname ) ; /* mutex */
ID porid = tkse_fnd_por ( ID domid, UB *oname ) ; /* rendezvous port */
```

Parameter

| | | |
|-----|-------|------------------|
| ID | domid | Kernel domain ID |
| UB* | oname | Object name |

Return Parameter

| | |
|---------|--|
| E_ID | Invalid ID number (domid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (object of oname does not exist) |
| E_PAR | Parameter error (oname is invalid or cannot be used) |

Description

Retrieves the object that belongs to the kernel or the process shown by domid by the object name, and gets the object ID. Specifies the object name of the object to be retrieved in oname.

In order to retrieve the object of other AMP TKSE, specify the kernel domain ID of the AMP TKSE in domid.

Objects with the global attribute of a specified AMP TKSE become the target of the retrieval.

If domid=TAR_SELF(=0) is specified, objects with the global attribute and kernel local attribute in its own AMP TKSE become the target of the retrieval.

If domid=TAR_PRIV(=-1) is specified, objects with the process local attribute of its own process become the target.

If the domain ID of its own AMP TKSE is specified, it is the same as when TAR_SELF is specified.

If the object specified in domid and oname is found, the ID of the object is returned. If the corresponding object does not exist, E_NOEXS is returned.

4.7 Standard Input/Output Function

4.7.1 Standard Input/Output Function Overview

Standard input/output of AMP TKSE mainly provides functions related to file input/output. The API specification of the AMP TKSE standard input/output function is the same as the T-Kernel Standard Extension Version 1.00 Specification.

The standard input/output function conducts file operations by using the following system calls in order to handle file systems with various specifications in a uniform manner unlike the standard file management function.

| | |
|---|--|
| ER tkse_attach(const TC *devnm, const char *connm, int mode) | file system connection |
| ER tkse_detach(const TC *devnm, int eject) | file system disconnection |
| ER tkse_open(const char *path, int oflag, mode_t mode) | open file system |
| ER tkse_close(int fildes) | close file system |
| ER tkse_lseek(int fildes, off_t offset, int whence) | move the current position of a file/directory |
| ER tkse_read(int fildes, void *buf, size_t nbyte) | read file |
| ER tkse_write(int fildes, const void *buf, size_t nbyte) | write file |
| ER tkse_getdents(int fildes, struct dirent *buf, size_t nbyte) | fetch directory entry |
| ER tkse_stat(const char *path, struct stat *sb) | retrieval of file information 1 |
| ER tkse_lstat(const char *path, struct stat *sb) | retrieval of file information 2 |
| ER tkse_fstat(int fildes, struct stat *sb) | retrieval of file information 3 |
| ER tkse_rename(const char *from, const char *to) | rename file |
| ER tkse_unlink(const char *path) | remove directory entry |
| ER tkse_mkdir(const char *path, mode_t mode) | directory creation |
| ER tkse_rmdir(const char *path) | directory removal |
| ER tkse_dup(int oldd) | file descriptor replication 1 |
| ER tkse_dup2(int oldd, int newd) | file descriptor replication 2 |
| ER tkse_fsync(int fildes) | file's disk cache content and disk synchronization |
| ER tkse_chdir(const char *path) | modify current directory 1 |
| ER tkse_fchdir(int fildes) | modify current directory 2 |
| ER tkse_chmod(const char *path, mode_t mode) | modify file mode 1 |
| ER tkse_fchmod(int fildes, mode_t mode) | modify file mode 2 |
| ER tkse_creat(const char *path, mode_t mode) | file creation |
| ER tkse_utimes(const char *path, const struct timeval times[2]) | modify access time, modification time |
| ER tkse_umask(mode_t cmask) | set file creation mask |
| ER tkse_truncate(const char *path, off_t length) | set file size to the specified length 1 |
| ER tkse_ftruncate(int fildes, off_t length) | set file size to the specified length 2 |
| ER tkse_sync(void) | disk cache content and disk synchronization |
| ER tkse_getfsstat(struct statfs *buf, W bufsize, int flags) | retrieve a list of file systems |
| ER tkse_getlink(const char *path, char *buf) | retrieve a LINK to standard file |

4.7.2 Target File System

File systems that can be handled by standard input/output are the following four kinds.

(1) T-Kernel Standard File System

Files and directories are not distinguished in the standard file system. On the standard input/output, they are classified as either a file or a directory according to the following conditions:

- directory

Files which include link records.

Indicates this directory and parent directories whose file type (file's application type) are 6 (cabinet pictogram)

- file

Files other than directory.

* The file's destination is one leading record only whose record type is 31. The target record is fixed when `tkse_open` or `tkse_write` is called first time after file creation, and remains the same until `tkse_close` is called.

(2) FAT File System

Accommodates FAT12, FAT16, FAT32 file system. Accommodates VFAT long file name.

It is accessible to both disks without partition information such as floppy disks and disks with partition information such as hard disks. However, for partition information, only basic partitions are supported.

(3) CD-ROM File System

Accommodates ISO 9660 Level 1 file system. Read only and unable to write.

(4) Shared Filesystem

Shared filesystems are shared by filesystem sharing management. For details, refer to "4.9 Filesystem Sharing Management".

4.7.3 File Access

(1) attach/detach file system

First, in order to access the file system on the device using STDIO, it is necessary to attach the file system(tkse_attach). The name of the file system specified at this time is called the "connection name".

To cancel connection, it is necessary to detach the file system(tkse_detach).

(2) open/close file

Writing and reading of the file become possible by opening of the file after file system connection. Success of opening of the file will assign a new file descriptor. File descriptor is the identifier with zero or more integral values, and performs the file operation using this identifier.

By closing of the file (tkse_close), the file descriptor becomes invalid.

The file descriptor is effective only within the process which opened the applicable file. File operation cannot be performed using the file descriptor which other processes opened.

All the files that the process opened are closed by the process termination.

A directory can also obtain the file descriptor by opening/closing.

4.7.4 Initial State of File Descriptor

The following file descriptors will be automatically opened at process startup:

| | | |
|---------------|---|-----------------------|
| STDIN_FILENO | 0 | standard Input |
| STDOUT_FILENO | 1 | standard output |
| STDERR_FILENO | 2 | standard error output |

These are all console I/Os assigned to the invoked process.

They cannot be closed unlike regular file descriptors.

4.7.5 Disk Cache

Using the disk cache, writing and reading can be performed efficiently. The data of the disk cache can be made to reflect in the file on a device by performing Close of the file, or the synchronization (tkse_sync, tkse_fsync) of the disk cache.

4.7.6 File Name

Directories and files cannot have the names which Japanese EUC (Including ASCII code) cannot describe. Note that directories and files with such names cannot be created. If there exist directories and files which have the names which Japanese EUC cannot describe, the results of retrieving the names are uncertain.

The maximum length of each file name may differ according to target file systems, any portion that exceeds the maximum length in each file system will be ignored.

- When referring a file:
The matching file name is found after any part greater than the maximum length is ignored.
- When creating a file:
The file is created with the name after any part greater than the maximum length is truncated.
- retrieval of file name
The file name is retrieved after any part greater than the maximum length is truncated.

* The following unique specification is applied only when the T-Kernel standard file system is used.

(1) Conversion to TRON code

In order to use TRON code for a file name in the T-Kernel standard file system, Japanese EUC is converted to TRON code inside the standard input/output.

In conversion from Japanese EUC to TRON code, single-byte characters are converted to corresponding double-byte characters (JIS level-1).

The conversion from TRON code to Japanese EUC is the reverse, and if there exist corresponding single-byte characters, they are converted to single-byte characters. However, two characters "/" and ":" are not converted to single-byte characters to distinguish path name and a delimiter representing the order in which it appears.

For example, file name string "Example1" is retrieved from the file called "Example1". When specifying this file, since either "Example1" or "Example1" is converted to "Example1", the same file will be specified.

In file names such as "Manuscript paper/E1", "/" is treated as a delimiter of path names, so the file called "Manuscript paper/E1" may not be specified. In this cases, the file may be specified by specifying "Manuscript paper/E1".

(2) Maximum filename length

File names up to 20 characters are allowed in the T-Kernel standard file system, but the maximum number of file names consisting of ASCII characters only will be extended to 34 characters by employing special encoding of file names.

Only if a file name is greater than 20 characters and it consists of ASCII characters (but characters only in the range 0x20-0x7e), the following special encoding will apply. Besides this, a file name is only converted from Japanese EUC to TRON code.

- Special encoding

Each character used in file names in the T-Kernel standard file system is encoded with two bytes (TC type = TRON code). Leading three characters of this file name consisting of two bytes/character are assumed to be a start mark of special encoding, 34 bytes of the remaining 17 characters make up a file name encoded with one byte/character.

- Start mark of special encoding

TK_U(0x2355), TK_X(0x2358), 0xA121

- File name encoding

ASCII codes (0x20-0x7e) are converted to 0x80-0xde and two characters each are packed and converted to TC type (two bytes/character format).

$$((c1 + 0x60) \ll 8) | (c2 + 0x60) \rightarrow \text{TC type}$$

c1: odd number character

c2: even number character

If the number of characters of a file name is odd, last one character will not be packed and be assumed to be a normal TRON code (TC type).

* 0xA121 corresponds to 1-1 code of JIS X 0212 (supplemental kanji set). This is undefined in JIS code and is usually not used.

* After packing, they are either undefined characters in D zone of TRON code or characters corresponding to KSC5601 (Korean).

4.7.7 Path Name

The path name is a character string that shows the location of the file. The character string of the path name sequences the route of the directory tree until it reaches the corresponding file from the root directory which is the starting point.

/connection name/directory name/file name

Example: "/CD/DIR_1/FILE.EXT"

The character code is Japanese EUC. The delimiter codes "/", "." may be single-byte characters (ASCII).

In T-Kernel standard file system, specification in the order it appears such as ":1" is available.

Example: "/SYS/DIR:1/FILE.EXT:2"

":" and numerals may be single-byte characters (ASCII).

* If all characters after ":" are numerals, the order they appear is specified. And, to find a file which already exists, use the file name with the order in which it appears, and to create a new file, use the file name without the order it appears.

For example, to create a file called "NEW_FILE:3", find NEW_FILE with the order in which it appears as "3". If it does not exist, create a file called "NEW_FILE".

* Since current directory function is not implemented, full path name should always be specified.

4.7.8 Root Directory

The root directory is a virtual directory with the highest rank among all other directories. The entity of the root directory does not exist on the file system.

Immediately under the root directory, all file systems currently connected exist as a virtual subdirectory. The name of each subdirectory becomes the connection name of the file system. When the file system is connected, a new virtual subdirectory is created under the root directory, and actual files and the directory are arranged under this virtual subdirectory.

The root directory is a directory for read only, and normal files and directories cannot be created immediately under the root directory. Moreover, it is not possible to change the name of a virtual subdirectory immediately under the root directory (connection name) by using the file name change system call.

For root directory, you can open and close the directory, fetch the directory entry, and retrieve directory information. You can also move current directory to root directory. If root directory is specified in other system calls, an error occurs.

4.7.9 Current Directory

A process maintains each individual current directory information. The current directory is used to realize file and directory access with the relative path name.

Current directory information of the process is succeeded by the child process. Moreover, the current directory of the initial process becomes the root directory "/" in the initial state.

Directories in which any process is set in the current directory cannot be deleted. However, the directory name can be changed.

4.7.10 This Directory "." and Parent Directory ".."

Its own directory "." and parent directory ".." can be used as the path name.

Its own directory displays the current directory, and the parent directory displays the directory which is one rank above. However, as an exception, the parent directory of the root directory displays the root directory itself.

4.7.11 Error Code

In libraries which the leverage standard input/output system calls, the following error codes are set to a variable `errno` when an error occurs:

```
#include <errno.h>
```

| | |
|---------|----------------------------------|
| EFAULT | illegal address |
| EINVAL | illegal parameter |
| ENOMEM | insufficient memory |
| EEXIST | already exist(s) |
| EMFILE | maximum open files exceeded |
| ESRCH | no process |
| EINTR | interrupted by a system call |
| EBADF | illegal file descriptor |
| EACCES | no access privileges |
| EPERM | processing not allowed |
| EROFS | unwritable file system |
| EXDEV | not the same file system |
| ENOENT | no file or directory |
| ENOSPC | insufficient disk space |
| ENODEV | processing on device not allowed |
| EIO | input/output error |
| EDEADLK | abnormal lock |
| EBUSY | busy |
| < 0 | other error |

4.7.12 System Calls

Attach file system

tkse_attach

C Language Interface

```
ER ercd = tkse_attach( const TC *devnm, const char *connm, int mode );
```

Parameter

| | | |
|------------|--------|---|
| const TC | *devnm | device name |
| const char | *connm | connection name |
| int | mode | connection mode (SF_STDFS SF_FATFS SF_CDROM) [SF_RDONLY] SF_RDONLY0x0001 read only SF_STDFS0x0000 T-Kernel Standard File system SF_FATFS0x0100 FAT File system SF_CDROM0x0200 CD-ROM File system |

Return Parameter

| | | | |
|----|------|-----|-------------------|
| ER | ercd | = 0 | normal completion |
| | | < 0 | error code |

Description

Connects the file system of a device with the device name of devnm by using the connection name of connm and connection mode of mode.

Connection names are 16 bytes or less (excluding '¥0' at the end of the character string). However, it becomes eight characters or less (in the case of 1 byte characters, 8 bytes or less, and in the case of 2 byte characters only, 16 bytes or less) for the T-Kernel standard file system.

The connection mode "mode" specifies the kind of connected file system.

T-Kernel standard file system/FAT file system/CD-ROM file system are each connected respectively by specifying SF_STDFS/SF_FATFS/SF_CDROM.

When SF_RDONLY is specified in the mode in addition to the kind of file system, the file system is connected as read only.

Detach file system

tkse_detach

C Language Interface

```
ER tkse_detach( const TC *devnm, int eject );
```

Parameter

| | | | |
|----------|--------|-----|--|
| const TC | *devnm | | device name |
| int | eject | | eject specification |
| | | = 0 | No eject |
| | | !=0 | Eject (Ignores devices in which eject is possible) |

Return Parameter

| | | | |
|----|------|-----|-------------------|
| ER | ercd | = 0 | normal completion |
| | | < 0 | error code |

Description

Detaches the file systems of devices with device name devnm.

When a file in the filesystem has been opened, detach is not possible.

Everything is synchronized when data exists on the disk cache.

Open File/Directory

tkse_open

C Language Interface

```
ER ercd = tkse_open( const char *path, int oflag, mode_t mode );
```

Parameter

| | | |
|------------|-------|----------------------------------|
| const char | *path | file path to open |
| int | oflag | open mode of file/directory |
| mode_t | mode | mode when "O_CREAT" is specified |

Return Parameter

| | | | |
|----|------|------|-------------------------------------|
| ER | ercd | >= 0 | normal completion (file descriptor) |
| | | < 0 | error code |

Description

Open the file/directory specified by the path name "path" in the open mode "oflag".

If successful, file descriptor (>=0) is returned to a return parameter.

The file descriptor is the minimum among the all unused numbers.

```
oflag := (O_RDONLY || O_WRONLY || O_RDWR) | [O_CREAT | [O_EXCL]] | [O_TRUNC] | [O_APPEND]
```

"oflag" can be one of the following:

| | | |
|----------|--------|------------|
| O_RDONLY | 0x0000 | read only |
| O_WRONLY | 0x0001 | write only |
| O_RDWR | 0x0002 | read/write |

Optionally, "oflag" can also be OR of the above and the following:

| | | |
|----------|--------|--|
| O_CREAT | 0x0200 | Create the file if there is not a file |
| O_TRUNC | 0x0400 | Delete file content |
| O_EXCL | 0x0800 | An error occurs if there is a file |
| O_APPEND | 0x0008 | Constantly appended at the end |

| | |
|----------|---|
| O_CREAT | Create the file if there is not a file. If the file already exists, the flag has no effect. Create a file in the mode specified by “mode”. |
| O_EXCL | Specify this along with “O_CREAT”. If the file already exists, an error occurs. Ignore if O_CREAT is not specified. |
| O_TRUNC | Discard file content and set file size to zero. Ignored if set to directory. Ignored in the case of read-only open (“O_RDONLY” specification). |
| O_APPEND | Constantly appended at the end of file when writing to the file. At this point, the current position is moved to the end. Same as moving to the end of file by using “tkse_lseek” just before “tkse_write”. |

“mode” should be specified only when “O_CREAT” is specified.

Specify the “mode” by taking the union of the followings with OR:

| | |
|---------------|----------------------------|
| S_IRWXU 00700 | owner RWX mask |
| S_IRUSR 00400 | owner R read permission |
| S_IWUSR 00200 | owner W write permission |
| S_IXUSR 00100 | owner X execute permission |
| S_IRWXG 00070 | group RWX mask |
| S_IRGRP 00040 | group R read permission |
| S_IWGRP 00020 | group W write permission |
| S_IXGRP 00010 | group X execute permission |
| S_IRWXO 00007 | other RWX mask |
| S_IROTH 00004 | other R read permission |
| S_IWOTH 00002 | other W write permission |
| S_IXOTH 00001 | other X execute permission |
| S_ISUID 04000 | run time user ID setting |
| S_ISGID 02000 | run time group ID setting |
| S_ISVTX 01000 | sticky bit |

These “mode” specifications have different scopes according to the target file systems. Invalid specifications will be ignored in the target file system.

In addition, the mask with “umask” will not be executed since the “umask” function is not currently implemented.

- T-Kernel Standard File system

The file type determined at “tkse_open()” time remains unchanged until the execution of “tkse_close()”. For example, when the file including link record is opened at “tkse_open()” as a directory, its file type is held as a directory even if link record is entirely deleted by other processes until the execution of “tkse_close()”

Set the file access attributes to read-only when read-only permission is given to its owner, group, and the others.

In other cases, write permission is given.

The file access mode is always set to default.

- FAT File system

The read-only attribute is set when read-only permission is given to the owner, group, and others.

In other cases, write permission is given.

Close file / directory

tkse_close

C Language Interface

```
ER ercd = tkse_close( int fildes );
```

Parameter

| | | |
|-----|--------|-----------------|
| int | fildes | file descriptor |
|-----|--------|-----------------|

Return Parameter

| | | | |
|----|------|-----|-------------------|
| ER | ercd | = 0 | normal completion |
| | | < 0 | error code |

Description

Closes the file/directory specified by the file descriptor “fildes”.

When a file which is open for writing is closed, if the same file is not open for writing at the same time in other locations, disk cache synchronization processing of the file (processing of writing back the content of the change from the disk cache to the disk) is conducted.

Move the current position of a file/directory

tkse_lseek

C Language Interface

```
ER ercd = tkse_lseek( int fildes, off_t offset, int whence );
```

Parameter

| | | |
|-------|------------|---|
| int | fildes | file descriptor |
| off_t | offset | offset from the specified position |
| int | whence | specify to start whence (SEEK_SET SEEK_CUR SEEK_END) |
| | SEEK_SET 0 | move to the "offset" position |
| | SEEK_CUR 1 | move to the current position + offset |
| | SEEK_END 2 | move to the end + offset |

Return Parameter

| | | | |
|----|------|------|---|
| ER | ercd | >= 0 | normal completion (Current position following the move) |
| | | < 0 | error code |

Description

Moves the current position (position in bytes) of the file/directory specified by the file descriptor "fildes".

When "fildes" designates a directory, "tkse_lseek" should not be used for purposes other than those of setting the current position to zero.

Read file**tkse_read****C Language Interface**

```
ER ercd = tkse_read ( int fildes, void *buf, size_t nbyte );
```

Parameter

| | | |
|--------|--------|----------------------|
| int | fildes | file descriptor |
| void | *buf | read buffer |
| size_t | nbyte | read size (in bytes) |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Reads “nbyte” bytes from the current position of the file specified by the file descriptor “fildes” to “buf”. Advance the file’s current position for the number of read bytes.

The number of bytes that succeeded in reading is returned as a return parameter. When the current position of the file is file terminal, 0 is returned. When an error occurs during reading, the error code is returned. In this case, the current position of the file is not changed.

Write file

tkse_write

C Language Interface

```
ER ercd = tkse_write( int fildes, const void *buf, size_t nbyte );
```

Parameter

| | | |
|-----------|--------|--------------------|
| const int | fildes | file descriptor |
| void | *buf | write-buffer |
| size_t | nbyte | write size (bytes) |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Writes “buf” from the current position of the file specified by the file descriptor “fildes” to “nbyte” bytes. Advance the file's current position for the number of written bytes.

The return parameter zero indicates the end of file. When an error occurs during writing, it is indeterminate how much data is written, but the file's current position is unchanged. When the current position of the file exceeds the actual file size, the behavior is as follows.

- T-Kernel Standard File system
An error occurs.
- FAT File system
The byte sequence of zero is written from the end of the file to the current position.
The number of bytes of the return parameter is not included in this area.

Get directory entry

tkse_getdents

C Language Interface

```
ER ercd = tkse_getdents( int fildes, struct dirent *buf, size_t nbyte );
```

Parameter

| | | |
|---------------|--------|--------------------------------|
| int | fildes | file descriptor |
| struct dirent | *buf | read buffer of directory entry |
| size_t | nbyte | read size (bytes) |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Reads directory entry (record) from the current position of the directory specified by the file descriptor “fildes”, and convert it into a “struct dirent” format to write to “buf”.

Specifies the size of “buf” in “nbyte” (bytes). Read contiguous directory entries to be entered into this size. Move the directory's current position so that the directory entry next to the directory entry that is lastly read can be indicated.

```
struct dirent {
    unsigned int    d_fileno;        /* file number */
    unsigned short d_reclen;        /* record length (the number of bytes) */
    unsigned char  d_type;          /* file type */
    unsigned char  d_namlen;        /* string length of "d_name" */
    char           d_name[255+1];    /* file name */
};
```

file type:

| | |
|------------|-------------------------------|
| DT_UNKNOWN | 0 unknown |
| DT_FIFO | 1 named pipe (FIFO) |
| DT_CHR | 2 character type special file |
| DT_DIR | 4 directory |
| DT_BLK | 6 block type special file |

| | |
|-----------|------------------|
| DT_REG | 8 normal file |
| DT_LNK | 10 symbolic link |
| DT_SOCKET | 12 socket |

“struct dirent” is a variable length data, and its size can be determined from “d_reclen”. When multiple directory entries are read, the position in “buf” where the next directory entry was stored can be determined by “d_reclen”.

The return parameter zero indicates the end of directory.

The number of bytes that succeeded in reading is returned as a return parameter. When the current position of the file is file terminal, 0 is returned. When an error occurs during reading, the error code is returned. In this case, the current position of the file is not changed.

- T-Kernel Standard File system

The file name including the order of appearance is stored in “d_name”. When directory entries are obtained by “tkse_getdents” multiple times, the order in which they appear may become incorrect by the accesses from other processes.

Get file information

tkse_stat

C Language Interface

```
ER ercd = tkse_stat ( const char *path, struct stat *sb );
```

Parameter

| | | |
|-------------|-------|--------------------------------|
| const char | *path | file path name |
| struct stat | *sb | buffer to get file information |

Return Parameter

| | | |
|----|------|-----------------------|
| ER | ercd | = 0 Normal completion |
| | | < 0 Error code |

Description

Get the information of the file specified by the path name “path” to store in “sb”.

```
struct stat {
    dev_t          st_dev; /* device ID */
    ino_t          st_ino; /* file serial number */
    mode_t        st_mode; /* file mode */
    nlink_t       st_nlink; /* the number of links */
    uid_t         st_uid; /* owner ID */
    gid_t         st_gid; /* group ID */
    dev_t         st_rdev; /* device type */
    struct timespec st_atimespec; /* latest access time */
    struct timespec st_mtimespec; /* latest update time */
    struct timespec st_ctimespec; /* latest file state update time */
    off_t         st_size; /* file size (bytes) */
    int64_t       st_blocks; /* the number of assigned blocks of file */
    u_int32_t     st_blksize; /* block size (the number of bytes) */
    u_int32_t     st_flags; /* user-defined flag */
    u_int32_t     st_gen; /* file generate number */
    int32_t       st_lspare; /* (reserved) */
    int64_t       st_qspare[2]; /* (reserved) */
};
```

```

#define st_atime      st_atimespec.tv_sec
#define st_mtime      st_mtimespec.tv_sec
#define st_ctime      st_ctimespec.tv_sec

#define S_BLKSIZE     512 /* block size (bytes) as an unit of "st_blocks */

```

The union (OR) value of the following values is returned to the file mode "st_mode"

```

#define S_IRWXU       0000700 /* owner RWX mask */
#define S_IRUSR       0000400 /* owner R read permission */
#define S_IWUSR       0000200 /* owner W write permission */
#define S_IXUSR       0000100 /* owner X execute permission */
#define S_IRWXG       0000070 /* group RWX mask */
#define S_IRGRP       0000040 /* group R read permission */
#define S_IWGRP       0000020 /* group W write permission */
#define S_IXGRP       0000010 /* group X execute permission */
#define S_IRWXO       0000007 /* other RWX mask */
#define S_IROTH       0000004 /* other R read permission */
#define S_IWOTH       0000002 /* other W write permission */
#define S_IXOTH       0000001 /* other X execute permission */
#define S_ISUID       0004000 /* run time user ID setting */
#define S_ISGID       0002000 /* run time group ID setting */
#define S_ISVTX       0001000 /* sticky bit */
#define S_IFMT        0170000 /* file type mask */
#define S_IFIFO       0010000 /* named pipe (FIFO) */
#define S_IFCHR       0020000 /* character type special file */
#define S_IFDIR       0040000 /* directory */
#define S_IFBLK       0060000 /* block type special file */
#define S_IFREG       0100000 /* normal file */
#define S_IFLNK       0120000 /* symbolic link */
#define S_IFSOCK      0140000 /* socket */

```

The user defined flag st_flags returns the logical sum of the following values.

```

#define SF_ARCHIVED 0x00010000 /* archive file */
#define SF_SYSTEM   0x40000000 /* system file */
#define SF_HIDDEN   0x80000000 /* hidden file */

```

```

struct timespec {

```

```

    time_t tv_sec;    /* seconds */
    long   tv_nsec;  /* nano seconds */
};

```

The number of seconds starting from the date and time at 00:00:00 GMT, Jan 1, 1985 shall be set to “time_t”. (It is based on TRON specifications, and is different from UNIX.)

When the time recorded in a file is prior to the standard date and time, zero (tv_sec=0, tv_nsec=0) is returned. When the time recorded in a file is beyond the time range designated by “time_t”, 0x7fffffff (tv_sec=0x7fffffff, tv_nsec=0) is returned.

The time recorded in a file is updated at the timing specified in the File system.

The information to be obtained may differ according to target File systems.

- T-Kernel Standard File system

st_dev device ID

Since device IDs are dynamically assigned when devices are registered, they are not fixed values.

“st_ino” file ID

“st_mode”, “S_IRUSR”, “S_IRGRP” and “S_IROTH” are constantly set.

“S_IXUSR”, “S_IXGRP”, and “S_IXOTH” are constantly set.

“S_IWUSR”, “S_IWGRP”, and “S_IWOTH” are set only when the read-only attribute is not set.

Owner, group, and other independent attributes are not set. It is always the same.

For the file type, either “S_IFDIR” or “S_IFREG” is set.

The file type at opening is set by “tkse_fstat”, and other attributes are determined according to the current file type or with or without a link record.

Other attributes are never set.

| | |
|--------------|-------------------------------|
| st_nlink | the number of file references |
| st_uid | (always 0) |
| st_gid | (always 0) |
| st_rdev | (always 0) |
| st_atimespec | latest access time |
| st_mtimespec | latest update time |
| st_ctimespec | file create time |
| st_size | target record size |

Only the size of record targeted for access is taken into account. Therefore, it will be smaller than the actual file size when multiple data records are included.

When the file type of the target file is a directory, the number of link records is set.

| | |
|------------|---|
| st_blocks | total number of used blocks |
| | the number of used blocks including total records and management information. |
| st_blksize | logical block size |
| st_flags | "SF_HIDDEN" is set for hidden virtual object. |
| st_gen | (always 0) |

- FAT File system

st_dev device ID

Since device IDs are dynamically assigned when devices are registered, they are not fixed values.

| | |
|--------------|--|
| st_ino | value based on the position of directory entry in the disk not always a fixed value. |
| st_mode | "S_IRUSR", "S_IRGRP", and "S_IROTH" are constantly set.. "S_IXUSR", "S_IXGRP", and "S_IXOTH" are constantly set. "S_IWUSR", "S_IWGRP", and "S_IWOTH" are set only when the read-only Owner, group, and other independent attributes are not set. It is always the same. For the file type, either "S_IFDIR" or "S_IFREG" is set. Other attributes are never set. |
| st_nlink | (always 1) |
| st_uid | (always 0) |
| st_gid | (always 0) |
| st_rdev | (always 0) |
| st_atimespec | latest access date (time is constantly 00:00:00) |
| st_mtimespec | latest update time |
| st_ctimespec | file create time access time and creation time are only set to VFAT. In other cases, update time is set to every attribute. |
| st_size | file size |
| st_blocks | the number of used blocks |
| st_blksize | cluster size |
| st_flags | "SF_ARCHIVED", "SF_SYSTEM", and "SF_HIDDEN" are set according to FAT file types. |
| st_gen | (always 0) |

- CD-ROM File system

st_dev device ID

Since device IDs are dynamically assigned when devices are registered, they are not fixed values.

| | |
|--------------|---|
| st_ino | value based on the position of directory record in the disk |
| st_mode | “S_IRUSR”, “S_IRGRP”, and “S_IROTH” are constantly set. “S_IXUSR”, “S_IXGRP”, and “S_IXOTH” are constantly set. For the file type, either “S_IFDIR” or “S_IFREG” is set. Other attributes are never set. |
| st_nlink | (always 1) |
| st_uid | (always 0) |
| st_gid | (always 0) |
| st_rdev | (always 0) |
| st_atimespec | date and time for recording |
| st_mtimespec | date and time for recording |
| st_ctimespec | date and time for recording |
| st_size | file size |
| st_blocks | the number of blocks to be used |
| st_blksize | logical block size |
| st_flags | “SF_HIDDEN” is set for a hidden file. |
| st_gen | (always 1) |

Get file information

tkse_lstat

C Language Interface

```
ER ercd = tkse_lstat( const char *path, struct stat *sb );
```

Parameter

| | | |
|-------------|-------|--------------------------------|
| const char | *path | file path name |
| struct stat | *sb | buffer to get file information |

Return Parameter

| | | |
|----|------|-----------------------|
| ER | ercd | = 0 normal completion |
| | | < 0 error code |

Description

Gets the information of the file specified by the path name “path” to store in “sb”.

Since the standard input/output function does not support symbolic link, the behavior of tkse_lstat() is the same as tkse_stat().

Get file information

tkse_fstat

C Language Interface

```
ER ercd = tkse_fstat( int fildes, struct stat *sb );
```

Parameter

| | | |
|-------------|--------|--------------------------------|
| int | fildes | file descriptor |
| struct stat | *sb | buffer to get file information |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Gets the information of the file specified by the path name “path” to store in “sb”.

Not only normal files but the console I/O (standard input/output) descriptor can also be specified.

- Console I/O (standard input/output)

| | |
|--------------|--|
| st_dev | (always 0) |
| st_ino | (always 0) |
| st_mode | “S_IRUSR S_IWUSR S_IFCHR” are set. (Fixed value) |
| st_nlink | (always 1) |
| st_uid | (always 0) |
| st_gid | (always 0) |
| st_rdev | (always 0) |
| st_atimespec | (always 0) |
| st_mtimespec | (always 0) |
| st_ctimespec | (always 0) |
| st_size | (always 0) |
| st_blocks | (always 0) |
| st_blksize | (always 0) |
| st_flags | (always 0) |
| st_gen | (always 0) |

Rename file

tkse_rename

C Language Interface

```
ER ercd = tkse_rename ( const char *from, const char *to );
```

Parameter

| | | |
|------------|-------|---------------------------|
| const char | *from | file name before changing |
| const char | *to | file name after changing |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Renames the file name 'from' to the file name 'to'.

When 'to' already exists, 'to' is deleted. In this case, 'from' and 'to' should be the same type (both are files or directories).

When 'from' and 'to' are in different directories, the files are moved between directories.

'from' and 'to' must be in the same file system. If 'from' and 'to' exist on different file systems, an error is returned.

- T-Kernel Standard file system

When the read-only attribute is set to the file indicated by 'to', an error occurs.

The file path name indicated by 'to' must be in the unopen state.

In the case 'from' is included in the path name 'to' or 'to' is a subdirectory of 'from' when renaming a directory, the directory is renamed.

* If 'from' is included in the path name 'to', files may not be accessed by using the path name hereafter.

- FAT File system

'to' must be in the unopen state.

In the case 'from' is included in the path name 'to' when renaming a directory, an error occurs.

Unlink directory entry

tkse_unlink

C Language Interface

```
ER ercd = tkse_unlink( const char *path );
```

Parameter

| | | |
|------------|-------|------------------------------|
| const char | *path | directory path to be deleted |
|------------|-------|------------------------------|

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Deletes the file specified by the path name “path”.

A directory or an open file cannot be deleted.

- T-Kernel Standard File system

In the case file type is other than six, discriminate file type depending on whether a link record exists at the invoked time or not. Therefore, an error occurs if the file type is six or the file includes a link record.

Make directory

tkse_mkdir

C Language Interface

```
ER ercd = tkse_mkdir( const char *path, mode_t mode );
```

Parameter

| | | |
|------------|-------|---|
| const char | *path | directory name to create |
| mode_t | mode | directory create mode * same as the "mode" of "tkse_open()". |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Makes the directory specified by the path name "path".

The directory name "." and ".." cannot be made.

Remove directory

tkse_rmdir

C Language Interface

```
ER ercd = tkse_rmdir( const char *path );
```

Parameter

| | | |
|------------|-------|--------------------------|
| const char | *path | directory name to remove |
|------------|-------|--------------------------|

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Removes the directory specified by the path name "path".

The directory to be removed (except "." and "..") must be blank.

The directory "." and ".." cannot be removed. The directory must be in the unopen state.

- T-Kernel Standard File system

Discriminates file type depending on with or without a link record at the invoked time.

Therefore, files without link records are targeted for deletion regardless of the file type.

Replicate File Descriptor

tkse_dup

C Language Interface

```
ER ercd = tkse_dup( int oldd );
```

Parameter

int oldd file descriptor to replicate

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Replicates the file descriptor "oldd" and returns a new file descriptor.

The file descriptor becomes the minimum number among numbers not being used.

The replicated file descriptor is handled as the same descriptor as "oldd" and the current location pointer is shared.

Replicate File Descriptor

tkse_dup2

C Language Interface

```
ER ercd = tkse_dup2( int oldd, int newd );
```

Parameter

| | |
|----------|------------------------------|
| int oldd | file descriptor to replicate |
| int newd | any new file descriptor |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Replicates the file descriptor "oldd" as a new file descriptor.

The replicated file descriptor is handled as the same descriptor as "oldd" and the current location pointer is shared.

If newd has already been used, close the file first, and then replicate.

File's Disk Cache Content and Disk Synchronization

tkse_fsync

C Language Interface

```
ER ercd = tkse_fsync( int fildes );
```

Parameter

| | | |
|-----|--------|-----------------|
| int | fildes | file descriptor |
|-----|--------|-----------------|

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Writes back data which has not been written back (not synchronized) from the disk cache to the disk among the writing operations to the file descriptor "fildes".

Returns after writing to the disk is complete.

Modify Current Directory

tkse_chdir

C Language Interface

```
ER ercd = tkse_chdir( const B *path );
```

Parameter

const B *path directory path to be changed

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error exit |

Description

Changes the current directory (working directory) to the directory indicated by the path name "path".

Modify Current Directory

tkse_fchdir

C Language Interface

```
ER ercd = tkse_fchdir( int fildes );
```

Parameter

int fildes file descriptor

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Changes the current directory (working directory) to the directory that has been opened as file descriptor "fildes".

Change File Mode

tkse_chmod

C Language Interface

```
#include <extension/seio.h>
```

```
ER tkse_chmod( const B *path, mode_t mode );
```

Parameter

| | | |
|---------|-------|-------------------------|
| const B | *path | file and directory path |
| mode_t | mode | mode specification |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Changes the mode of the file or the directory specified by the path name "path".

Mode is a logical sum of the following values.

| | | |
|---------------|-------------------|----------------------|
| S_IRWXU 00700 | owner | RWX mask |
| S_IRUSR 00400 | owner | R read permission |
| S_IWUSR 00200 | owner | W write permission |
| S_IXUSR 00100 | owner | X execute permission |
| S_IRWXG 00070 | group | RWX mask |
| S_IRGRP 00040 | group | R read permission |
| S_IWGRP 00020 | group | W write permission |
| S_IXGRP 00010 | group | X execute permission |
| S_IRWXO 00007 | other | RWX mask |
| S_IROTH 00004 | other | R read permission |
| S_IWOTH 00002 | other | W write permission |
| S_IXOTH 00001 | other | X execute permission |
| S_ISUID 04000 | run time user ID | setting |
| S_ISGID 02000 | run time group ID | setting |
| S_ISVTX 01000 | sticky bit | |

These mode settings have different scopes according to the target file systems. Invalid settings will be ignored in the target file system.

If the file modes of already opened files are changed, the changes here will not affect them until they are closed.

Change File Mode

tkse_fchmod

C Language Interface

```
ER ercd = tkse_fchmod( int fildes, mode_t mode );
```

Parameter

| | | |
|--------|--------|--------------------|
| int | fildes | file descriptor |
| mode_t | mode | mode specification |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error code |

Description

Changes the mode of the file or the directory that has been opened as file descriptor "fildes".

As for the mode specification, the valid range differs according to targeted file system. Invalid specifications are ignored in the targeted file system.

When the file modes of already opened files are changed, the changes will not affect them until they are closed.

Create File

tkse_creat

C Language Interface

```
ER ercd = tkse_creat( const B *path, mode_t mode );
```

Parameter

| | | |
|---------|-------|-------------------------|
| const B | *path | file and directory path |
| mode_t | mode | mode specification |

Return Parameter

| | | | |
|----|------|------|---|
| ER | ercd | >= 0 | normal completion (file descriptor of created file) |
| | | < 0 | error exit |

Description

Creates and opens files with path name "path".

The file descriptor as a return value is the minimum of all unused numbers.

Performs a processing equivalent to the setting (O_CREAT | O_WRONLY | O_TRUNC) to open's oflag.

Modify Access Time, Modification Time

tkse_utimes

C Language Interface

```
ER ercd = tkse_utimes( const B *path, const struct timeval times[2] );
```

Parameter

| | | |
|----------------------|----------|---------------------------------------|
| const B | *path | file and directory path |
| const struct timeval | times[2] | modify access time, modification time |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error exit |

Description

Changes the access time, the modification time of a file/directory to which the path name path points.

utimes sets times[0].tv_sec to access time, times[1].tv_sec to modification time.

Sets seconds since the date and time at 00:00:00 GMT, Jan 1, 1985 to time_t, timeval.tv_sec. (It is based on TRON specifications, and is different from UNIX.)

If NULL is set to times, file's access time and modification time are set to current time.

- Standard File system

If zero is set to access time and modification time, these times are unchanged.

```
struct timeval {
    long tv_sec;        /* second */
    long tv_usec;      /* microsecond */
};
```

Set File Creation Mask

tkse_umask

C Language Interface

```
mode_t tkse_umask( mode_t cmask );
```

Parameter

mode_t cmask mask value

Return Parameter

ER ercd \geq 0 normal completion (Mask value prior to setting)
 < 0 error code

Description

Sets the file creation mask value of its own process in cmask.

The value specified in mask value "cmask" is removed from the mode value "mode" specified when the file is created. This value is the mode value which is applied when the file is created.

cmask is a logical sum of the following values..

| | |
|---------------|----------------------------|
| S_IRWXU 00700 | owner RWX mask |
| S_IRUSR 00400 | owner R read permission |
| S_IWUSR 00200 | owner W write permission |
| S_IXUSR 00100 | owner X execute permission |
| S_IRWXG 00070 | group RWX mask |
| S_IRGRP 00040 | group R read permission |
| S_IWGRP 00020 | group W write permission |
| S_IXGRP 00010 | group X execute permission |
| S_IRWXO 00007 | other RWX mask |
| S_IROTH 00004 | other R read permission |
| S_IWOTH 00002 | other W write permission |
| S_IXOTH 00001 | other X execute permission |
| S_ISUID 04000 | run time user ID setting |
| S_ISGID 02000 | run time group ID setting |
| S_ISVTX 01000 | sticky bit |

These umask settings have different scopes according to the target file systems. Invalid settings will be ignored

in the target file system.

The initial value of system's cmask is set to zero, and the process inherits parent process's cmask.

Set File Size to the Specified Length

tkse_truncate

C Language Interface

```
ER ercd = tkse_truncate( const B *path, off_t length );
```

Parameter

| | | |
|---------|--------|-------------------------|
| const B | *path | file and directory path |
| off_t | length | file size to specify |

Return Parameter

| | | | |
|----|------|------|-------------------|
| ER | ercd | >= 0 | normal completion |
| | | < 0 | error |

Description

Extends or truncates the file size of a file to which path name path points or the file size of a file opened by files to the specified size length bytes.

If length is less than the file size, the file size will be truncated to length bytes, and truncated portion will be lost.

If length is greater than the file size, the file size will be extended to length bytes, and zero is written in the extended portion.

If length is equal to the file size, nothing is done.

Set File Size to the Specified Length

tkse_ftruncate

C Language Interface

```
ER ercd = tkse_ftruncate( int fildes, off_t length );
```

Parameter

| | | |
|-------|--------|----------------------|
| int | fildes | file descriptor |
| off_t | length | file size to specify |

Return Parameter

| | | | |
|----|------|------|---|
| ER | ercd | >= 0 | normal completion (file descriptor of created file) |
| | | < 0 | error exit |

Description

Extends or truncates the file size of a file to which path name path points or the file size of a file opened by fildes to the specified size length bytes.

If length is less than the file size, the file size will be truncated to length bytes, and truncated portion will be lost.

If length is greater than the file size, the file size will be extended to length bytes, and zero is written in the extended portion.

If length is equal to the file size, nothing is done.

Disk Cache Content and Disk Synchronization

tkse_sync

C Language Interface

```
void tkse_sync( void );
```

Parameter

none

Return Parameter

none

Error Code

none

Description

Writes all the not yet written data in memory to disk.

Returns after writing to disk is complete.

Retrieve a List of File systems

tkse_getfsstat

C Language Interface

```
ER ercd = tkse_getfsstat( struct statfs *buf, W bufsize, int flags );
```

Parameter

| | | |
|---------------|---------|------------------------------------|
| struct statfs | *buf | retrieval information storage area |
| W | bufsize | storage area size |
| int | flags | flags (unused) |

Return Parameter

| | | | |
|----|------|------|---|
| ER | ercd | >= 0 | normal completion (file descriptor of created file) |
| | | < 0 | error exit |

Description

Retrieves information about all the connected file systems to store it in buf.

bufsize is the size (number of bytes) of buf's area, all the information that can be stored in buf concerning file system. For example, if bufsize = sizeof(struct statfs) * 10, information concerning up to ten file systems will be stored. Information on the number of file systems that were successfully acquired is returned as a return value. If NULL is set to buf, bufsize will be ignored and the number of connected file systems will be returned as a return value.

flags is an argument reserved for future extensions.. Always specify MNT_WAIT (= 1).

Since the root file system is a virtual file system, file system information cannot be acquired.

```
typedef struct fsid {
    W val[2];
} fsid_t;          /* file system ID */

#define MNAMELEN 90 /* maximum length of connection name/device name */

struct statfs {
    W      f_spare2;          /* (blank) */
    W      f_bsize;          /* logical block size (B: number of bytes) */
    W      f_iosize;         /* optimal block size to transfer (B) */
```



```

W      f_blocks;          /* file system space (LB: number of logical blocks) */
W      f_bfree;          /* file system free space (LB) */
W      f_bavail;        /* free space available to general users (LB) */
W      f_files;         /* * maximum number of files */
W      f_ffree;         /* * number of blank files */
fsid_t f_fsid;          /* file system ID (always 0) */
uid_t  f_owner;         /* connected user (always zero) */
int    f_type;          /* file system type */
int    f_flags;         /* connect flag */
W      f_spare[6];      /* (blank) */
B      f_mntonname[MNAMELEN]; /* connection name */
B      f_mntfromname[MNAMELEN]; /* device name */
};

```

* The value of items marked with * may be undefined according to file systems. In such cases, the items will be set to -1.

In the current implementation, free space `f_bavail` available to general users is equal to file system free space `f_bfree`.

The file system type `f_type` is one of the following:

```

#define MOUNT_FATDS    4      /* FAT File system */
#define MOUNT_CDFS    14     /* CD-ROM File system */
#define MOUNT_STDFS   20     /* Standard File system */
#define MOUNT_SFS     42     /* Shared File System */

```

The value of connect flag `f_flags` is set to the result of taking the union of the following information:

```

#define MNT_RDONLY    0x00000001 /* read-only */

```

The connection name `f_mntonname` is a path name from root.

Example: `"/SYS"`

The device name `f_mntfromname` is set to the device name pretended by `"/dev/"`.

Example: `"/dev/pca0"`

Retrieve a LINK to Standard File

tkse_getlink

C Language Interface

```
ER ercd = tkse_getlink( const B *path, B *buf );
```

Parameter

| | | |
|---------|-------|------------------------------------|
| const B | *path | file path of standard input/output |
| B | *buf | LINK information storage area |

Return Parameter

| | | | |
|----|------|------|---|
| ER | ercd | >= 0 | normal completion (file descriptor of created file) |
| | | < 0 | error exit |

Description

Returns LINK information based on standard file system specifications of a file or directory denoted by the path name path to buf.

buf should be an area with the size greater than or equal to sizeof(LINK).

LINK information is stored to buf and zero is returned as a return value, only if the file or directory specified by path is a file in standard file system. If the file or directory specified by path is not a file in standard file system. The content of buf is indefinite and an error is returned.

LINK information is retrievable without access privileges to the file or directory specified by path. However, access privileges to the target file or directories which is included in path is needed just like the access privileges to directories would be required when a file is opened with a file is opened with tkse_open().

4.8 Standard File Management Function

4.8.1 Standard File Management Function Overview

The standard file management function provides the standard file system of “AMP TKSE” and the functions to manipulate its files.

“AMP TKSE” recommends that normal file manipulation should be executed by using the standard input/output functions. If you wish to use the functions specific to standard file system, the standard file management functions shall be used.

The standard file system has a structure based on real/virtual object models, with following features:

- File organization consisting of ordered record sequence with variable length (record stream)
- Any network-like reference relationships via links (virtual object) included in files
(A directory in the traditional file system does not exist.)
- Direct access to files via links (virtual object)

Since files are used by multiple users in chronological order and furthermore used simultaneously by multiple users in network environment, detailed file access management and high-level protection mechanism are provided.

However, the current version of “AMP TKSE” does not support multiple users.

The API specification of the AMP TKSE standard file management function is the same as the T-Kernel Standard Extension Version 1.00 Specification.

4.8.2 File and Link

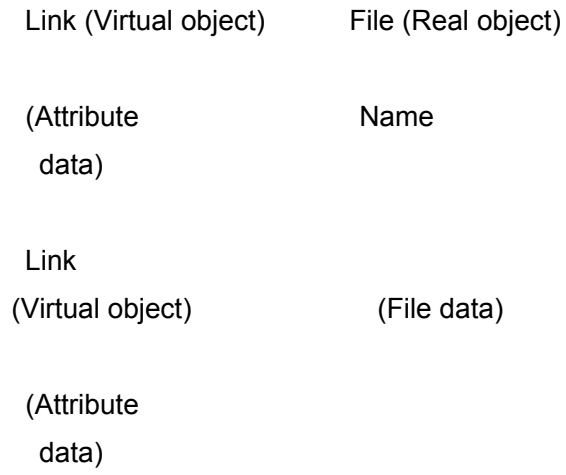
A file consists of ordered record sequence with variable length.

A link is a kind of key pointer for referring to a file, and it has a data structure composed of data which indicates referred file and several attribute data specific to the link.

The link can exist as a record with being embedded in any files. Multiple links indicating a file can exist, and consequently network-like reference relationships among any files can be defined as a whole.

In the correspondence to real/virtual object model, a file has one-on-one correspondence to a real object and a link has one-on-one correspondence to a virtual object.

In general, a file is directly referred via the link. Therefore, a file name does not have absolute meaning and is used as a search key. A file can have any file name of up to 20 characters, and it doesn't matter if multiple files with the identical file name exist.



[Figure 9] Relation of file and links

4.8.3 File System

A file system is a physical unit for managing files, is built in storage media, and has a ceiling on physical size.

The file system certainly has a root file, and all files in the file system are reachable by sequentially following the links included in the root file.

In the correspondence to real/virtual object model, a root file corresponds to a device real object.

In general, reference relationships between files via links are defined and integrated in a single file system, but an indirect link for referring to files in other file system is available and particularly called an indirect link. Because a reference via the indirect link is not integrated with changes in other destination file system, be aware that the existence of destination file is not assured.

The file system name and the device location name are set when the file system is generated.

A file system name is a name of up to 20 characters that is also set as a root file name and used by the system and users to absolutely discriminate file systems. A file system name must be unique, because file systems with identical file system name are regarded as the identical.

The device location name consists of up to 20 characters that indicate a physical device where a file system is stored. And it is used to access other machines via the network or to ask the installation of floppy, etc.

4.8.4 Connect File System

There is no file system available at system startup, a file system will become available just after the connection operation. Therefore, minimum connection of file system is normally required as system initialization.

A file system is connected by specifying the logical device name where destination file system exists and the connection name. The connection name is a name of up to 8 characters for discriminating a connected file system, and is used as an absolute path name that indicates a root file of connected file system. When connected, a link to the root file of the connected file system is also obtained.

Therefore, use of link or connection name obtained at connection enables the access to a root file of the connected file system. The access to a root file of the connected file system is enabled by using link or connection name obtained at connection, and sequential tracing of the following links from a root file allows you to access any file in the connected file system. The access to any file in the connected file system is enabled by following links from a root file.

A file system is disconnected by specifying the logical device name of the file system targeted to be disconnected. Consequently, a file access via the link which indicates a file in disconnected file system is unavailable. This state is called a disconnected state.

In the correspondence to real/virtual object model, a link in disconnected state corresponds to a virtual object.

The connection of file system is a function simply to register the existence of file system dynamically with the file management function, and is a flat connection without any structure. Therefore, network-like static file reference structure across multiple file systems is built by using indirect links which refer to files in the different file systems.

4.8.5 File ID

At file generation, a unique number called file ID is attached to every file in the file system to be internally discriminated. A file ID is a value in the range from 0 (maximum file ID), and the maximum file ID (namely, maximum file count) is defined at file system generation. Since the file ID is represented by 16 bit numeric value, the maximum file ID can not exceed 65535.

A root file in the file system constantly has the file ID 0.

4.8.6 Link

A link is a kind of key pointer for accessing a file, and it has a data structure which holds file system name where destination file exists, file ID, and several attribute data as links.

The link is simply a dynamic data as a pointer, but it becomes a fixed existence by being stored as a record in a file. Thus, the stored link is particularly called a fixed link. Since the fixed link does not have file system name, only the reference to the file in the same file system is possible. And when fixed link is taken out from the file, the file system name to which the file belongs is set as a data structure of the link.

Thus, to store a link which refers to different file system in a file as a fixed link, a special file called a link file shall be generated in the file system to be stored in advance, and a link in it which indicates the link file shall be stored as a fixed file.

A link file is a special file which holds file system name where the file to be referred exists, file ID, file name, and generation time and date. And the access to the link file is automatically interpreted as an access to a file in the different destination file system. A link which indicates a link file is particularly called an indirect link, and a link which indicates a normal file is called a direct link.

Since a multiple indirect link, namely reference to file via more than two link files, is not supported, an error indicating that the file does not exist occurs when accessed.

A file reference via the link file by an indirect link is as follows:

- Identify a file system by its file system name. An access is disabled when unconnected. In addition, a connection name is irrelevant to the access via a link file.
- Check the file in the file system identified by file ID. And when both or either of file name and generation time

matches and the file is not a link file, accesses the file as a target. In other cases, the file to be referred is regarded as nonexistent and the access is disabled.

4.8.7 Working File

The file currently targeted for processing by certain process is called a working file of the process. A process enables any file to be a working file by using system call.

A working file is held as an execution environment in a process, and is inherited to generated child processes.

The working file can be undefined, and the working file of a process first generated in a system is in undefined state.

4.8.8 Path Name

In general, a file is directly referred by a link. However, a link cannot be interactively traced in a batch-style application, etc, so a file can be referred by specifying link sequence to be directly traced.

As a link sequence for this purpose, a list in order of file names referred by each link is called a path name. In this case, only a file name does not assure the uniqueness. So, a file name shall be used by appending it with the order of appearance.

The order it appears is a serial number assigned from 0 to (n-1) when there are “n” number of links referring to files with the identical file name in a file. When the order it appears is omitted, it is assumed to be zero, namely the first time.

File XYZ

(1) File ABC

(2)

(3) File ABC

The reference to link (1) is ABC or ABC:0

The reference to link (2) is ABC:1

The reference to link (3) is ABC:2

[Figure 10] Example of the order of appearance in pathname

The path name has the following syntax and is treated as one character-string of up to 256 characters:

```
[Path name]           ::= [Special reference][Special reference] /
                        [Simple path name][Simple path name]
[Simple path name]    ::= [Simple path name] / [Reference specification]
                        [Reference specification]
[Reference specification] ::= [File name] [[File name]:[Order it appears]]
[Special reference]   ::= / [Connection name]
[Order it appears]    ::= Numeric value
[File name]           ::= String (up to 20 characters)
[Connection name]    ::= String (up to 8 characters)
```

A special reference has the following meaning:

```
/ [Connection name] -- indicating the root file of the file system connected by the specified
                        connection name.
-- indicating a working file.
```

As the symbols “/”, “:”, “.” are special codes as follows, displayable all characters including blank(space) is enabled as the file name. If “/” is existent at the end of path name character-string, it will be ignored.

```
/      TC_FDLM 0xff21
:      TC_FSEP 0xff22
      TC_FOWN 0xff23
```

The path name beginning with “/ [Connection name]” is a path name from the root file of a file system, and is called an absolute path name. In other cases, a path name is a relative path name from the current working file, and is called a relative path name.

A path name can be indicated as follows:

```
/latest/project/software specifications/core specification/file management
external specification/chapter 10/example:1
```


4.8.9 File Type

There are mainly two types of files as follows, and a file simply means a normal file.

Normal file: a file in a usual sense in terms of the place where data is stored.

Link file : a special file used to indirectly refer to files in another file system, and an indirect link is the link indicating this type of file.

4.8.10 Normal File Composition

A normal file is composed of an ordered sequence of any byte-length records, namely record streams, and each record is composed of the following elements:

- Record type
- Record subtype
- Record size
- Record body

The record type is a value in the range of 0 to 31 which indicates the type of a record.

0 Link record

A record which stores a link to the other file. Since the content is directly treated by file management function, direct manipulation of it from applications is restricted.

1-31 Data record

A record type defined as a system. File management function, however, has no concern with its content and treats it as just a byte sequence.

The record subtype is an auxiliary type specification used according to the record type and a 16 bit unsigned numeric value used for a keyword.

The record size is 32 bit data that indicates the number of bytes of a record body. Although the link record does not have record size information, the size (52 bytes) of LINK structure, which indicates the size of an area required for input/output of records, is set to the record size. However, the size of this link record is not counted in the total number of bytes as file management information.

The record body is a data sequence of the number of bytes specified by the record size, and its content is determined depending on the record type. The record body of the link record is specially treated.

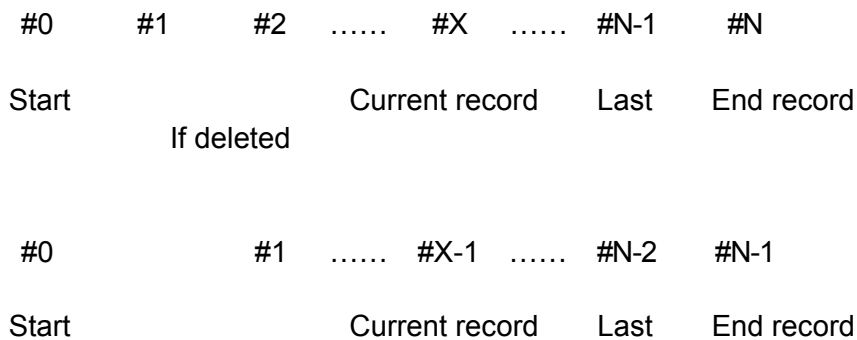
4.8.11 Record Number / Current Record

Each record of a file is numbered in sequence according to the order of record defining the first record as “0”, and this number is called a record number. As the record number indicates the order of records, it dynamically changes by record insertion/deletion.

A record next to the last record is deemed to virtually exist, and this record is called the end record. If there are “N” pieces of records, the end record will have record number “N”.

A current record is defined as a target record to be currently accessed in the opened files, and a data access is executed to the record of the current record. A current record can be moved by specifying the record number, and a search based on the record type, etc.

The current record is not changed even by record insertion/deletion, and only the record number corresponding to the current record is changed.



[Figure 11] Change of the record number by record deletion

4.8.12 Link File Composition

A link file is a file which is generated and used to indirectly refer to a file in the different file systems. In the link file, there is not any application data, but only the following management data is stored:

- File ID of the file to be referred
- Application type of the file to be referred
- File name of the file to be referred
- Generation date and time of the file to be referred
- File system name where the file to be referred exists
- Device location name of the file system where the file to be referred exists

4.8.13 File Control

A file is accessed by processes. A positive integer called file descriptor ($fd > 0$) defined by each process is assigned to the open file, and the actual file is accessed using this file descriptor.

On process termination, all the opened files are automatically closed. A current record is also defined as a target record to be accessed for the opened file.

The file descriptor and the current record location are defined as specific to the process, and are not especially passed on to child processes.

A working file is passed on to child processes as a process environment.

4.8.14 Reference Count of File

In a file, a reference count which indicates the number of fixed links referring to the file in the same file system exists. The reference count is zero at file generation, and is incremented by one when a fixed link to the file is generated, namely a link is stored in the file. Conversely, the reference count is decremented by one when fixed link is deleted.

As the reference count indicates references in the same file system, file references via link file are not reflected in the reference count. In addition, the reference count is applied to the link file itself, too.

A file deletion is enabled only for a file with reference count zero. If a fixed link is included in the deleted file, the reference count of the file to which the fixed link refers is decremented by one. And even if it results in zero, the file will not be deleted. Meanwhile, a file deletion which includes a fixed link is enabled only when forced deletion is specified at deletion.

The same holds for link file deletion, and it is enabled when the reference count of the link file itself is zero. Note that the destination file of the link file cannot be deleted via link file.

The reference count of the root file in the file system is exceptionally one from the beginning, so the way it works that it cannot be deleted at all.

A file with reference count zero does not have a fixed link which refers to it. Therefore, when dynamic link is lost, it cannot be accessed in the usual way. However, it can be accessed by retrieving links to all files in the file system.

4.8.15 File Access

A file is to be opened by specifying any of “READ”/“WRITE”/“UPDATE”, and the following mode specification is enabled in order to restrict the simultaneous open of the same file from others at opening: The mode setting defaults to share mode, but usually an exclusive write mode is a safe option.

Exclusive mode: prohibit any simultaneous open from others

Exclusive write mode: prohibit simultaneous open for writing/updating from others.

Share mode: not prohibit any simultaneous open from others.

The combination of the mode that enables a new simultaneous open to the mode that has been already opened is as shown below. If newly simultaneous open is not enabled, an error occurs at opening.

[Table 1] Simultaneous Open Mode Combinations

| Existing open mode | | Newly simultaneous open mode | | | | | | | | |
|----------------------|---|------------------------------|---|---|-----------------|---|---|-------|---|---|
| | | Exclusive | | | Exclusive write | | | Share | | |
| | | R | W | U | R | W | U | R | W | U |
| Exclusive mode | R | x | x | x | x | x | x | x | x | x |
| | W | x | x | x | x | x | x | x | x | x |
| | U | x | x | x | x | x | x | x | x | x |
| Exclusive write mode | R | x | x | x | | x | x | | x | x |
| | W | x | x | x | x | x | x | | x | x |
| | U | x | x | x | x | x | x | | x | x |
| Share mode | R | x | x | x | | | | | | |
| | W | x | x | x | x | x | x | | | |
| | U | x | x | x | x | x | x | | | |

A record lock function to prohibit others from executing access to each record of the opened file is also prepared.

The accesses to the locked record from others are as follows:

- To read, write, replace, reduce size, and delete the record result in an error.
- To make the record search target or the current record is enabled.

When you try to lock an already locked record, an error occurs or you are forced to wait until it is unlocked.

4.8.16 File System Management Information

The following management information for each file system can be read:

```
typedef struct {
    UH    fs_bsize;           /* the number of bytes of logical block */
    UH    fs_nfile;          /* maximum number of files */
    H     fs_lang;           /* language used in the file system */
    H     fs_level;          /* access management level of the file system */
    W     fs_nblk;           /* total number of blocks */
    W     fs_nfree;          /* total number of unused blocks */
    STIME fs_mtime;          /* last updated time of the system block */
    STIME fs_ctime;          /* creation time of a file system */
    TC    fs_name[L_FSNM];   /* file system name */
    TC    fs_locat[L_DLNM];  /* device location name */
} FS_STATE;
```

- "fs_bsize" is the number of bytes of one logical block, and shall be the power of 2.
- "fs_nfile" indicates the maximum number of file registrable in the file system. This value equals to the maximum file ID + 1.
- "fs_lang" indicates the language used in the file system, and represents the character code system used in this file system.
- "fs_level" represents an access management level in the file system, and can be the following values:
 - 0: Level 0 -- no access management
 - 1: Level 1 -- access management (no hidden name)
 - 2: Level 2 -- access management (hidden name)
- "fs_nblk" indicates total number of logical blocks in the file system, and this value equals to maximum value of logical block number +1.
- "fs_nfree" is a current total number of unused logical blocks, and this data fluctuates dynamically.
- "fs_mtime" and "fs_ctime" are respectively the last updated time and the generated time of the file system represented by seconds since the reference date and time at starting from 00:00:00 GMT, Jan 1, 1985.
- "fs_name" and "fs_locat" are respectively the names set at generation time (at the time of format)of the file system, and it is padded with trailing zeros if the name is less than 20 characters in length.

The management information of the file system is set at generation time (format time) of the file system, and

thereafter unchanged except for total number of unused logical blocks (fs_nfree), last updated time of the system blocks (fs_mtime), file system name and device location name.

4.8.17 File Management Information

The following management information for each file can be read: However in case of the link file, the information of a file to which the link file refers will be read, and the management information of the link file itself can not be read.

File name:

A file name of 20 characters. It may be modified.

Reference count:

Indicate the number of fixed links referring to the file in the same file system.

File management information:

Various management information is as shown below:

```
typedef struct {
    UH    f_type;           /* file type/owner access mode */
    UH    f_atype;         /* application type */
    TC    f_owner[L_USRNM]; /* file owner name (in the case of hidden name, it is
                           constantly zero) */
    TC    f_group[L_USRNM]; /* owner group name (in the case of hidden name, it is
                           constantly zero) */
    UH    f_grpacc;        /* group access level */
    UH    f_pubacc;        /* general access level */
    H     f_nlink;         /* the number of included links */
    H     f_index;         /* index level */
    W     f_size;          /* total number of bytes of the file */
    W     f_nblk;          /* total number of used logical blocks */
    W     f_nrec;          /* total number of records */
    STIME f_ltime;         /* shelf life of the file (date and time) */
    STIME f_atime;         /* latest access time */
    STIME f_mtime;        /* last updated time */
    STIME f_ctime;         /* file create time */
} F_STATE;
```

• "f_type" indicates file type, access attribute, and owner access mode, as follows:

TTTT xxxx BAPO xRWE

T: file type

0 link file

1 normal file

2- reserved

P: permanent attribute

The value one indicates that this file is prohibited from removal.

O: read-only attribute

The value one indicates that this file is read-only.

A: application attribute 1

B: application attribute 2

The attribute specified and used by an application. The file management has no concern with its meaning.

RWE: file owner access mode (Respectively enabled in the case of 1)

x : reserved (zero)

- Application type (f_atype) is the data set and used by an application, and this data is not used by the file management.
- Owner name (f_owner) and owner group name (f_group) consist of 12 characters each; if it consists of less than 12 characters, it is padded with trailing zeros. The subsequent hidden name of two characters is always obtained as zeros.
- The group access level (f_grpacc) and the general access level (f_pubacc) have the following compositions:
 - xxxx RRRR WWWW EEEE
 - RRRR : lowest readable user level (0-15)
 - WWWW : lowest writable user level (0-15)
 - EEEE : lowest executable user level (0-15)
 - xxxx : unused (0)
- The number of included links (f_nlink) indicates the number of link records which the file includes.
- The index level indicates the indirect multiplicity of 0-based record index.
- The total number of bytes of the file (f_size) is the total number of bytes of data actually written in the file, and is the total amount of the record size of each record. In this case, the record size of the link record is counted as zero.
- The total number of logical blocks in use indicates the total number of logical blocks used in the file.
- The total number of records indicates the total number of records existent in the file.
- The date and time is set to the number of seconds starting from the date and time at 00:00:00 GMT (Greenwich Mean Time), Jan 1, 1985. This data is indicated to be invalid in the case the value is -1.

Latest access time (f_atime)

Time when the file data is last read or the index is last updated. At file generation, -1 (if not supported) or generation time of the file is set.

Last updated time (f_mtime)

The time when the file data is last updated. At file generation, the generation time is set.

File creation time (f_ctime)

The time when the file is generated for the first time.

Shelf life (f_ltime)

Shelf life of the file. -1 is set when file is generated. This data is set and used by an application. It is not used in the file management.

File location information:

The file system information to which each file belongs. This content is a part of the management information of the file system.

```
typedef struct {
    STIME    fs_ctime;           /* creation time of a file system */
    TC       fs_name[L_FSNM];   /* file system name */
    TC       fs_locat[L_DLNM];  /* device location name */
    TC       fs_dev[L_DEVNM];   /* logical device name */
} F_LOCATE;
```

- The logical device name is the name of the block type device where the file system exists at the point.

Link file information:

For the link file, the following destination information held in the link file itself can be obtained: This information can be retrieved even when the destination file system is not connected.

4.8.18 Link Structure

Used to access files. The link has following data structure:

```
typedef struct {
    TC       fs_name[L_FSNM]; /* file system name */
    UH       f_id;           /* file ID */
    UH       atr1;           /* Attribute data 1 */
    UH       atr2;           /* Attribute data 2 */
}
```



```
        UH      atr3; /* Attribute data 3 */
        UH      atr4; /* Attribute data 4 */
        UH      atr5; /* Attribute data 5 */
    } LINK;
```

- The file system name is the connected file system name itself and used to absolutely discriminate the file system. When it is set to fixed link, this information will not be stored in the file.
- The file ID is a file ID in the file system identified by the file system name.
- Attribute data 1-5 are attribute data held as link itself and their usage, and determined by upper level applications since the file management has no concern with their contents in general. All the default values of a newly generated link shall be zero. This data is stored in the file when it is set to fixed link, and the content stored in the file is retrieved when the fixed link is read.

In the file management, actual file access is executed only by using the file system name and the file ID.

In general, a link obtained from the file management function is used, but an application can create the link by directly setting the file system name and the file ID.

For example, as the file ID of a root file is zero, a direct link to the root file in the file system can be created by an application if the file system name is available.

4.8.19 System Calls

Get Link to File

tkse_get_Ink

C Language Interface

```
ER ercd = tkse_get_Ink(TC *path, LINK *Ink, W mode);
```

Parameter

| | | |
|------|-------|---|
| TC | *path | target path name NULL target is a working file |
| LINK | *Ink | storage area of obtained link (output) specify working file (input: when "F_BASED" is set) |
| W | mode | mode to get link (F_NORM F_BASED) [F_DIRECT] F_NORM specify normal file F_BASED specify working file F_DIRECT specify to get direct link |

Return Parameter

| | | |
|----|------|---|
| ER | ercd | < 0 error code 0 normal completion (a link to a normal file) 1 normal completion (a link to a link file: F_DIRECT not specified) 2 normal completion (a link to a normal file to which the link file refers: When "F_DIRECT" is set) |
|----|------|---|

Error Code

| | |
|----------|---|
| E_FACV | No route file access right (E) within the path name (path) |
| E_MACV | Address (path,Ink) access is not permitted |
| E_FNAME | Path name (path) is empty, invalid or too long. |
| E_IO | Input/output error occurred |
| E_NOFS | The file system to which the file within the path name (path) and the reference file of the link file (when F_DIRECT is specified) belong are not connected |
| E_NOEXS | The file within the path name (path) and the reference file of the link file (when F_DIRECT is specified) do not exist or working file is undefined |
| E_PAR | Parameter is invalid. (mode is invalid) |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Gets a link to the file specified by the path name.

When NULL is specified by the path name, get a link to the current working file.

When the path name is relative path name, with F_NORM specification, the link shall be based on the current working file, and with "F_BASED" specification, the link shall be based on the file specified by "lnk" as a working file.

When the specified file is a link file, without "F_DIRECT" specification, a link to the link file itself shall be obtained. In this case, the existence of the normal file to which obtained link file refers is not assured.

In the case of "F_DIRECT" specification, a direct link to the normal file to which the link file refers is obtained.

To retrieve the link to a file, an access privileges to execute/search (E) each file included in the path name is needed, but an access privileges to execute/search (E) the destination file itself is not needed.

Change Working File

tkse_chg_wrk

C Language Interface

```
ER ercd = tkse_chg_wrk(LINK *lnk);
```

Parameter

| | | |
|------|------|-------------------------------|
| LINK | *lnk | working file to be changed |
| | NULL | set working file as undefined |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|---|
| E_OK | Normal completion |
| E_FACV | No file (lnk) access right (E) |
| E_MACV | Address (lnk) access is not permitted |
| E_IO | Input/output error occurred |
| E_NOEXS | File (lnk) does not exist |
| E_NOFS | File system where the file (lnk) belongs is not connected |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Sets the specified file to a working file of invoking process.

To set a working file, an access privileges to execute/search (E) the file is required.

Create File

tkse_cre_fil

C Language Interface

```
ER ercd = tkse_cre_fil(LINK *lnk, TC *name, A_MODE *mode, UH atype, W opt);
```

Parameter

| | | |
|--------|-------|---|
| LINK | *lnk | storage area of the link to created file (output) specify file system (input: when "F_FLOAT" is set) specify parent file (input: when "F_FIX" is set) specify file to be created (input: when "F_FILEID" is set) |
| TC | *name | file name (valid for 0 or up to maximum number of file name characters) |
| A_MODE | *mode | access mode NULL apply default access mode |
| UH | atype | file application type |
| W | opt | attribute of creation (FLOAT F_FIX F_FILEID) F_FLOAT floating link specification F_FIX fixed link specification F_FILEID file ID specification |

Return Parameter

| | | |
|----|------|---|
| ER | ercd | < 0 error code > 0 normal completion (file descriptor) |
|----|------|---|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_FACV | No file (lnk) access right (W) (when F_FIX is specified) |
| E_MACV | Address (lnk,name,mode) access is not permitted |
| E_BUSY | Since file (lnk) has already been exclusively opened, the file could not be opened simultaneously (when F_FIX is specified) |
| E_OBJ | File (lnk) already exists (when F_FILEID is specified) |
| E_FNAME | File name (name) is empty or invalid |
| E_IO | Input/output error occurred |
| E_LIMIT | Maximum number of files exceeded or maximum number of files which can be opened simultaneously exceeded File (lnk) exceeded the maximum size of the system (when F_FIX is specified) |

| | |
|------------|--|
| E_NODSK | Disk area is insufficient |
| E_NOEXS | File (lnk) does not exist (when F_FIX is specified) |
| E_NOFS | File system where the file (lnk) belongs is not connected |
| E_PAR | Parameter is invalid (f_grpno<0,>4 ,opt is invalid) |
| E_RDONLY | File (lnk) is write-protected or file system that file belongs to is write-protected |
| E_SYSTEMEM | Memory area of the system is insufficient |

Description

Create normal new file and open it to update in the file system where the file (in the case of link file, a normal file to which the link file refers) specified by "lnk" exists.

All the attribute data of the link to the generated file are set to zero, and are stored in the area specified by "lnk".

In the case of "F_FLOAT" specification, a file shall be simply created. The reference count of the created file is set to 0.

In this case, since only the file system name specified by "lnk" is valid and file ID is ignored, the file specified by "lnk" doesn't need to exist.

In the case of "F_FIX" specification, a link to the created file is appended to the last record position of the file specified by "lnk" as a link record (subtype = 0) at the last record position of the file specified by "lnk". The reference count of the created file shall be set to 1. In this case, the file specified by "lnk" shall exist and be able to be write opened.

In the case of "F_FILEID" specification, create a file with the same file ID as the file ID specified by "lnk". The reference count of the created file is set to 0. In this case, the file specified by "lnk" shall not exist.

"A_MODE" specifies the access mode of created file.

Even when owner access mode of the created file is read-only, the file is opened to update, and the record number is set to zero.

Create Link File

tkse_cre_Ink

C Language Interface

```
ER ercd = tkse_cre_Ink(LINK *Ink, F_LINK *ref, W opt);
```

Parameter

| | | |
|--------|------|---|
| LINK | *Ink | storage area of the link to created file (output) specify file system (input: when "F_FLOAT" is specified) specify parent file (input: when "F_FIX" is specified) specify file to be created (input: when "F_FILEID" is specified) |
| F_LINK | *ref | link file content to be created |
| W | opt | content of link file to be created (F_FLOAT F_FIX F_FILEID) F_FLOAT floating link specification F_FIX fixed link specification F_FILEID file ID specification |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_FACV | No file (Ink) access right (W) (when F_FIX is specified) |
| E_MACV | Address (Ink,ref) access is not permitted |
| E_BUSY | Since file (Ink) has already been exclusively opened, the file could not be opened simultaneously (when F_FIX is specified) |
| E_OBJ | File (Ink) already exists (when F_FILEID is specified) |
| E_FNAME | File name (ref->f_name) or file system name (ref->fs_name) is empty or invalid |
| E_IO | Input/output error occurred |
| E_LIMIT | Maximum number of files exceeded File (Ink) exceeded the maximum size of the system (when F_FIX is specified) |
| E_NODSK | Disk area is insufficient |
| E_NOEXS | File (Ink) does not exist (when F_FIX is specified) |
| E_NOFS | File system where file (Ink) belongs is not connected |
| E_PAR | Parameter is invalid (opt is invalid, same file system) |

- E_RDONLY File (lnk) is write-protected or file system that file belongs to is write-protected (when F_FIX is specified)
- E_SYSMEM Memory area of the system is insufficient.

Description

Creates a link file with the content specified by "ref" in the file system where the file (In the case of link file, a normal file to which the link file refers) specified by "lnk" exists.

All the link attribute data of the link to the created file are set to zero, and are stored in the area specified by "lnk". The meaning of "F_FLOAT", "F_FIX", and "F_FILEID" is identical with "tkse_cre_fil()".

The content of the created link file is the one specified by "ref", but the creation time is set to the time when this system call is executed instead of "ref->f_ctime".

The actual existence of the file specified by "ref" is not checked.

When "ref->fs_name" is identical with the file system name specified by "lnk", a link file cannot be created. That results in an error.

Generate File Directly

tkse_gen_fil

C Language Interface

```
ER ercd = tkse_gen_fil(LINK *lnk, TC *name, F_STATE *stat, F_LINK *ref, W opt);
```

Parameter

| | | |
|---------|-------|--|
| LINK | *lnk | storage area of generated file link (output) specify file system (input: when "F_FLOAT" is specified) specify parent file (input: when "F_FIX" is specified) specify file to be generated (input: when "F_FILEID" is specified) |
| TC | *name | file name (valid for 0 or up to maximum number of file name characters) (valid only at normal file generation; when name is NULL at this time, an error occurs) (Not referred at all when generating a link to file) |
| F_STATE | *stat | file content to be generated |
| F_LINK | *ref | link file content to be generated (valid only at link file generation) |
| W | opt | attribute of generation (F_FLOAT F_FIX F_FILEID) F_FLOAT floating link specification F_FIX fixed link specification F_FILEID file ID specification |

Return Parameter

| | | |
|----|------|---|
| ER | ercd | < 0 error code = 0 normal completion (at link file generation) > 0 normal completion (file descriptor: at normal file generation) |
|----|------|---|

Error Code

| | |
|---------|---|
| E_FACV | Not level 0 user |
| E_MACV | Address (lnk,ref, name, stat) access is not permitted |
| E_BUSY | Since file (lnk) has already been exclusively opened, the file could not be opened simultaneously (when F_FIX is specified) |
| E_OBJ | File (lnk) already exists (when F_FILEID is specified) |
| E_FNAME | File name (name), file name (ref->f_name) and file system name (ref->fs_name) are empty or invalid |

| | |
|------------|---|
| E_IO | Input/output error occurred |
| E_LIMIT | Maximum number of files exceeded or maximum number of files which can be opened simultaneously exceeded |
| | File (lnk) exceeded the maximum size of the system (when F_FIX is specified) |
| E_NODSK | Disk area is insufficient |
| E_NOEXS | File (lnk) does not exist |
| E_NOFS | File system where the file (lnk) belongs is not connected |
| E_PAR | Parameter is invalid (opt is invalid, contents of same file system, ref, stat are invalid) |
| E_RDONLY | File (lnk) is write-protected or file system that file belongs to is write-protected |
| E_SYSTEMEM | Memory area of the system is insufficient |

Description

Newly generates a normal file or a link file in the file system where the file (In the case of link file, normal file to which the link file refers) specified by "lnk" exists, and open it for updating if a normal file is generated

All the attribute data of the link to the generated link file is set to zero, and is stored in an area specified by "lnk".

The meaning of "F_FLOAT", "F_FIX", and "F_FILEID" is identical with "tkse_cre_fil()".

The generated file content is specified by "stat", and whether it is a normal file or a link file is distinguished by "stat->f_type".

At normal file generation, a normal file with the name specified by "name" shall be generated and the generated file management information shall be set to the content specified by "stat". However, the values of "f_nlink", "f_index", "f_size", "f_nblk", and "f_nrec" are ignored and initialized at file generation.

All other contents of "stat" are ignored and link file of "ref" content is generated. It is same as the behavior of "tkse_cre_lnk()", and "ref->f_ctime" is valid as well.

Since this system call is used for special purpose of restoring a file system, etc., it can be executed only in a process at user level 0.

At normal file generation, the file is opened for updating. Since there is no record in this state, the current record indicates the end record and the record number is zero.

Open File

tkse_opn_fil

C Language Interface

```
ER ercd = tkse_opn_fil(LINK *lnk, W o_mode, TC *pwd);
```

Parameter

| | | |
|------|--------|--|
| LINK | *lnk | target file |
| W | o_mode | open mode (F_READ F_WRITE F_UPDATE) [F_EXCL F_WEXCL] F_READ open for reading F_WRITE open for writing F_UPDATE open for updating (reading/writing) F_EXCL exclusive mode F_WEXCL exclusive write mode |
| TC | *pwd | password NULL no password specification |

Return Parameter

| | | |
|----|------|---|
| ER | ercd | < 0 error code > 0 normal completion (file descriptor) |
|----|------|---|

Error Code

| | |
|----------|---|
| E_FACV | No file (lnk) access right (o mode is supported) |
| E_MACV | Address (lnk,pwd) access is not permitted. |
| E_BUSY | Since file (lnk) has already been exclusively opened, the file could not be opened simultaneously |
| E_IO | Input/output error occurred |
| E_LIMIT | Maximum number of files which can be opened simultaneously exceeded |
| E_NOEXS | File (lnk) does not exist |
| E_NOFS | File system where the file (lnk) belongs is not connected |
| E_PAR | Parameter is invalid (o mode is invalid) |
| E_PWD | The password of file (lnk) does not match |
| E_RDONLY | File (lnk) is write-protected or file system that file belongs to is write-protected |
| E_SYMEM | Memory area of the system is insufficient |

Description

Opens the file specified by “lnk” in the specified mode. To open a file, an access privileges corresponding to the open mode is required.

Since password function is currently unsupported, “pwd” is set to NULL.

The current record is set to the start record of the opened file. When there is no record in the file, the current record is set to the end record.

Close File

tkse_cls_fil

C Language Interface

```
ER ercd = tkse_cls_fil(W fd);
```

Parameter

| | | |
|---|----|-----------------|
| W | fd | file descriptor |
|---|----|-----------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------|--------------------------------|
| E_OK | Normal completion |
| E_FD | File descriptor does not exist |
| E_IO | Input/output error occurred |

Description

Closes the opened file.

When the process which opened the file exits, the file will be automatically closed.

Delete File

tkse_del_fil

C Language Interface

```
ER ercd = tkse_del_fil(LINK *org, LINK *lnk, W force);
```

Parameter

| | | |
|------|-------|--|
| LINK | *org | parent file of the file targeted to be deleted NULL no parent file specification |
| LINK | *lnk | file targeted to be deleted |
| W | force | forcible deletion specification = 0 : not deleted when the destination file includes a link record. != 0 : deleted when the destination file includes a link record. |

Return Parameter

| | | |
|----|------|--|
| ER | ercd | < 0 error code ≥ 0 normal completion (the number of link records when reference count results in zero after deletion) |
|----|------|--|

Error Code

| | |
|-----------------|--|
| E_FACV | No file (org) access rights (W) (when org!=NULL) |
| E_MACV | Address (org, lnk) access is not permitted |
| E_BUSY | Since file (org) has already been exclusively opened, the file could not be opened simultaneously (when org!=NULL) File (lnk) is being opened or is a working file The reference count of file (lnk) is not 0 (when org=NULL) The link record which displays file (lnk) is being used as current record by another open file. |
| (When org=NULL) | |
| E_IO | Input/output error occurred |
| E_LOCK | The link record which displays the file (lnk) is being locked by another link (when org!= NULL) |
| E_NOEXS | File (org, lnk) does not exist (or the link record which displays the file (lnk) specified within org does not exist) |
| E_NOFS | File system where the file (org, lnk) belongs is not connected |
| E_PERM | File (lnk) cannot be deleted (deletion impossible attribute is set) |
| E_REC | File contains link record. (when force=0) |
| E_RDONLY | File (org) cannot be written or the file system that the file belongs to cannot be written (when |

org!= NULL)

File system (Ink) that the file belongs to cannot be written

E_SYSMEM Memory area of the system is insufficient

Description

Delete the link record which indicates the file specified by “Ink” in the parent file specified by “org”, and decrement the reference count of the file by one. When reference count results in zero, the file itself specified by “Ink” shall be deleted. In this case, an access privileges to write (W) parent file is required.

In the case of no parent file setting (org = NULL), if the reference count of the file specified by “Ink” is zero, the file shall be deleted. An error is caused by the reference count other than zero.

When the file targeted for deletion is a link file, the link file itself is the target for deletion; the destination file for reference of the link file will not be deleted.

In the case of no forced deletion (force = 0), if the file to be deleted include a link record, the file will not be deleted as an error. In the case of forced deletion (force != 0), if the file to be deleted includes a link record, the file shall be deleted and the reference count of the file shall be decremented by one which the included link record indicates. And as the result, the number of link records whose reference count results in zero is set to return value.

When the file to be deleted is in any of the following, an error occurs without the deletion of the file:

- Permanent attribute is set
- open process is existent
- process which is used as a working file is existent

The file can be deleted even when read-only attribute is set.

Move Current Record

tkse_see_rec

C Language Interface

```
ER ercd = tkse_see_rec(W fd, W offset, W mode, W *recnum);
```

Parameter

| | | |
|---|---------|---|
| W | fd | file descriptor |
| W | offset | offset to move |
| W | mode | move mode |
| | = 0 | move to the record number for current record number + offset |
| | > 0 | move to the position for “offset” record number It should be offset >= 0 |
| | < 0 | move to the record number position for the end of record number + “offset” It should be offset <=0 |
| W | *recnum | storage area of the current record number after move NULL not stored |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|--------|--|
| E_OK | Normal completion |
| E_MACV | Address (recnum) access is not permitted |
| E_FD | File descriptor does not exist |
| E_IO | Input/output error occurred |
| E_REC | Range of the existing record exceeded |

Description

Moves the current record position for the opened file to the specified location.

When specified destination exceeds the existing range of record, an error occurs and the current record will not be changed.

Find Record

tkse_fnd_rec

C Language Interface

```
ER ercd = tkse_fnd_rec(W fd, W mode, UW typemask, UH subtype, W *recnum);
```

Parameter

| | | |
|----|----------|---|
| W | fd | file descriptor |
| W | mode | search mode (specify start position/direction to search) (F_FWD F_NFWD F_BWD F_NBWD F_TOPEND F_ENDTOP) F_FWD from the current record to the end record F_NFWD from the record next to the current one to the end of record F_BWD from the current record to the top record F_NBWD from the record previous to the current one to top record F_TOPEND from the top record to the end of record F_ENDTOP from the end of record to the top record |
| UW | typemask | “bitmask” of the record type targeted for search support for LSB type 0 support for MSB type 31 |
| UH | subtype | record subtype targeted for search 0 applied to all subtypes (without subtype check) |
| W | *recnum | storage area of the current record number as a result of search NULL not stored |

Return Parameter

| | | | |
|----|------|------|--|
| ER | ercd | < 0 | error code |
| | | >= 0 | normal completion (searched record type) |

Error Code

| | |
|--------|--|
| E_MACV | Address (recnum) access is not permitted |
| E_FD | File descriptor does not exist |
| E_IO | Input/output error occurred |
| E_PAR | Parameter is invalid (mode is invalid) |
| E_REC | Record matching to the specified retrieval conditions does not exist (including cases when typemask=0) |

Description

Searches the specified record in the opened file, and sets the found record as the current record.
When target record is not found, an error occurs and the current record will not be changed.

Find Link Record

tkse_fnd_Ink

C Language Interface

```
ER ercd = tkse_fnd_Ink(W fd, W mode, LINK *Ink, UH subtype, W *recnum);
```

Parameter

| | | |
|------|---------|---|
| W | fd | file descriptor |
| W | mode | search mode (specification of start position/direction/content to search) (F_FWD F_NFWD F_BWD F_NBWD F_TOPEND F_ENDTOP) [F_SFILE] [F_SNAME] [F_SATR1] [F_SATR2] [F_SATR3] [F_SATR4] [F_SATR5] same as F_FWD ~ F_ENDTOP tkse_fnd_rec() F_SFILE link record which indicates the same file as "Ink" F_SNAME link record which indicates the file with the same file name as "Ink" F_SATR1 link record with the same attribute data 1 as "Ink" F_SATR2 link record with the same attribute data 2 as "Ink" F_SATR3 link record with the same attribute data 3 as "Ink" F_SATR4 link record with the same attribute data 4 as "Ink" F_SATR5 link record with the same attribute data 5 as "Ink" |
| LINK | *Ink | targeted link for search enabled only when F_SFILE - F_SATR5 is specified |
| UH | subtype | targeted record subtype for search 0 applied to all subtypes (without subtype check) |
| W | *recnum | storage area of the current record number as a result NULL not stored |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|--------|---|
| E_OK | Normal completion |
| E_MACV | Address (Ink,recnum) access is not permitted (Ink is accessed only when the retrieval conditions are specified) |
| E_FD | File descriptor does not exist |

| | |
|----------|---|
| E_IO | Input/output error occurred |
| E_NOEXS | File (lnk) does not exist |
| E_NOFS | File system where file (lnk) belongs is not connected |
| E_PAR | Parameter is invalid (mode is invalid) |
| E_REC | Record matching the specified retrieval conditions does not exist |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Searches the specified link record in the opened file, and sets the found link record as the current record. When destination record is not found, an error occurs and the current record will not be changed.

Read Record

tkse_rea_rec

C Language Interface

```
ER ercd = tkse_rea_rec(W fd, W offset, B *buf, W size, W *r_size, UH *subtype);
```

Parameter

| | | |
|----|----------|---|
| W | fd | file descriptor |
| W | offset | byte position to start reading (>=0) |
| B | *buf | storage area of read data NULL not stored |
| W | size | byte size of read data storage area (>=0) |
| W | *r_size | remaining byte size from the starting byte position storage area with the size (record size - offset) NULL not stored |
| UH | *subtype | storage area of record type NULL not stored |

Return Parameter

| | | |
|----|------|--|
| ER | ercd | < 0 error code >= 0 normal completion (the record type of the current record) |
|----|------|--|

Error Code

| | |
|--------|---|
| E_MACV | Address (buf,r_size,subtype) access is not permitted |
| E_ENDR | Current record is an end record |
| E_FD | File descriptor does not exist or is F_WRITE open |
| E_IO | Input/output error occurred |
| E_LOCK | Current record is locked from others |
| E_PAR | Parameter is invalid (Offset and size are invalid in size<0, offset<0, and the link record) |

Description

Reads the current record of the opened file.

When record size < offset + size, only the (record size - offset) bytes data is read and stored in "buf".

When "offset >= record size", "buf = NULL", or "size = 0", nothing is stored in "buf" but values corresponding to "*r_size" and "*subtype" are stored. This is used when retrieving record type or subtype only.

When the current record is a link record, the content of the entire "LINK" structure is read to "buf", and the size of "LINK" structure is stored in "*r_size". In this case, the condition must be met that "offset = 0", "size >= size of "LINK" structure" (or size = 0).

When the current record is the end record, or is locked by other process, an error occurs.

Write Record

tkse_wri_rec

C Language Interface

```
ER ercd = tkse_wri_rec(W fd, W offset, B *buf, W size, W *r_size, UH *subtype, UW units);
```

Parameter

| | | |
|----|----------|---|
| W | fd | file descriptor |
| W | offset | byte position to start writing (-1 <= offset < record size) -1: write to the end of record (addition) |
| B | *buf | pointer to the write data NULL not written |
| W | size | byte size of write data (>=0) |
| W | *r_size | remaining byte size from the starting byte position storage area of the size (record size after writing - offset) NULL not stored |
| UH | *subtype | pointer to the record type to be modified NULL No change |
| UW | units | unit to get block (K bytes) 0 any |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_MACV | Address (buf,r_size,subtype) access is not permitted |
| E_ENDR | Current record is an end record |
| E_FD | File descriptor does not exist or is F_READ open |
| E_IO | Input/output error occurred |
| E_LOCK | Current record is locked from others |
| E_NODSK | Disk area is insufficient or specified continuous block area could not be acquired |
| E_PAR | Parameter is invalid (size<0, offset is invalid, and offset, size are invalid in the link record) |
| E_LIMIT | The maximum file size of the system was exceeded |

Description

Writes to the current record of the opened file.

When $\text{record size} < \text{offset} + \text{size}$, the record size increases after writing.

"units" specifies the unit of getting additional blocks in K bytes necessary for record size increase, and specifies to allocate consecutive block area of greater than or equal to "units" size (less than or equal to "size").

"units = 0" means that any way to allocate blocks is allowed.

When "size = 0" or "buf = NULL", data is not written. When "subtype != NULL", record subtype is changed.

When "buf = NULL" and " $\text{record size} < \text{offset} + \text{size}$ ", record size is increased. The data for(of) increased portion is indeterminate. This is used when reserving record's additional block area in combination with "units" setting.

When "offset = -1", data is constantly written to the end of record at this point, and the value of "size" is stored in "*r_size". Even when the same record is opened by multiple processes and simultaneously opened, this setting assures that written data is not overwritten by another data.

When the current record is a link record, the content of "buf" is "LINK" structure. However, only portion of attribute data is written, the file itself to be referred cannot be changed. In this case, the condition must be met that "offset = 0", "size >= size" of LINK structure (or "size = 0").

When the current record is the end record or the current record is locked by other process, an error occurs.

Insert Record

tkse_ins_rec

C Language Interface

```
ER ercd = tkse_ins_rec(W fd, B *buf, W size, W type, UH subtype, UW units);
```

Parameter

| | | |
|----|---------|--|
| W | fd | file descriptor |
| B | *buf | pointer to the inserted record data |
| | NULL | data is not written |
| W | size | byte size of the inserted record (>=0) |
| W | type | record type of inserted record |
| UH | subtype | subtype of the inserted record |
| UW | units | unit to get block (K bytes) |
| UH | subtype | record subtype of inserted record |
| | 0 | any |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_MACV | Address (buff) access is not permitted |
| E_FD | File descriptor does not exist or is F_READ open |
| E_IO | Input/output error occurred |
| E_LIMIT | The file in the reference link exceeded the maximum reference count (255) of the system The maximum file size of the system was exceeded |
| E_NODSK | Disk area is insufficient or specified continuous block area could not be acquired |
| E_NOEXS | The file at the reference link does not exist |
| E_PAR | Parameter is invalid (type is invalid, size<0 and units are invalid, and size and buff are invalid when type=0) |
| E_REC | Link is referring to a different file system |

Description

Inserts new record just before current record of the opened file.

"units" specifies the unit to get blocks in K bytes necessary in inserted record, and specifies to allocate consecutive block area of greater than or equal to "units" size (less than or equal to size). The "units = 0" means that any way to allocate blocks is allowed.

When "buf = NULL", the size of inserted record becomes "size", but the data is indeterminate. This is used to get record's block area beforehand in combination with "units" specification.

When "type = 0", a link record is inserted and the content of "buf" is set to the "LINK" structure. By inserting a link record, the reference count of the file which the link indicates is incremented by one. In this case, the condition must be met that "buf != NULL", "size = size of "LINK" structure", and the file which the link indicates must exist in the same file system.

Append Record

tkse_apd_rec

C Language Interface

```
ER ercd = tkse_apd_rec(W fd, B *buf, W size, W type, UH subtype, UW units);
```

Parameter

| | | |
|----|---------|---|
| W | fd | file descriptor |
| B | *buf | pointer to the additional record data NULL data is not written |
| W | size | byte size of the additional record (>=0) |
| W | type | record type of additional record |
| U | subtype | record subtype of the additional record |
| UW | units | unit to get block (K bytes) 0 any |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_OK | Normal completion |
| E_MACV | Address (buff) access is not permitted |
| E_FD | File descriptor does not exist or is F_READ open |
| E_IO | Input/output error occurred |
| E_LIMIT | The file in the reference link exceeded the maximum reference count (255) of the system The maximum file size of the system was exceeded |
| E_NODSK | Disk area is insufficient or specified continuous block area could not be acquired |
| E_NOEXS | The file at the reference link does not exist |
| E_PAR | Parameter is invalid (type is invalid, size<0 and units are invalid, and size and buff when type =0 are invalid) |
| E_REC | Link is referring to a different file system |

Description

Inserts new record into the end of the opened file.

This system call is identical with "tkse_ins_rec()" excepting that a record is constantly inserted into the last record

(before the end of record) regardless of the position of current record.

Delete Record

tkse_del_rec

C Language Interface

```
ER ercd = tkse_del_rec(W fd);
```

Parameter

W fd file descriptor

Return Parameter

ER ercd < 0 error code
 = 1 normal completion (deletion of link record results in reference count = 0)
 = 0 normal completion (other than the mentioned above)

Error Code

E_BUSY It is being used as current record by another open file
 E_ENDR Current record is an end record
 E_FD File descriptor does not exist or is F_READ open
 E_IO Input/output error occurred
 E_LOCK Current record is locked from others
 E_SYSMEM Memory area of the system is insufficient

Description

Deletes the current record of the opened file and moves the current record to the next record of the deleted one. When the deleted record is a link record, the reference count of the file which the link record indicates is decremented by one. When reference count then results in zero, return value one is returned.

An error occurs when the current record is the end record, is locked by other process, or is the current record of other open process.

Truncate Record Size

tkse_trc_rec

C Language Interface

```
ER ercd = tkse_trc_rec(W fd, W size);
```

Parameter

| | | |
|---|------|--------------------------------------|
| W | fd | file descriptor |
| W | size | record byte size to be reduced (>=0) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|--------|--|
| E_OK | Normal completion |
| E_ENDR | Current record is an end record |
| E_FD | File descriptor does not exist or is F_READ open |
| E_IO | Input/output error occurred |
| E_LOCK | Current record is locked from others |
| E_PAR | Parameter is invalid (size<0) |
| E_REC | Current record is a link record |

Description

Truncates record size of the current record of the opened file to "size" bytes. Nothing shall be executed when record size <= "size".

An Error occurs when current record is the end record, link record, or locked by other processes.

Exchange File Content

tkse_xch_fil

C Language Interface

```
ER ercd = tkse_xch_fil(W fd_1, W fd_2);
```

Parameter

| | | |
|---|------|-------------------|
| W | fd_1 | file descriptor 1 |
| W | fd_2 | file descriptor 2 |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|---|
| E_OK | Normal completion |
| E_FD | File descriptor does not exist or is not (F_UPDATE F_EXCL). It is open. |
| E_IO | Input/output error occurred |
| E_LOCK | Current record is locked from others (record is locked) |
| E_PAR | Parameter is invalid (fd_1 and fd_2 are the same file) |
| E_XFS | File (fd_1, fd_2) belongs to a different file system |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Exchanges the contents of the opened two files.

Only data part of file are exchanged; the file management information remains unchanged except access date and update time.

The two files to be exchanged must exist in the same file system and must be opened for update in the exclusive mode.

The current records after exchange are respectively the top record.

Record lock

tkse_loc_rec

C Language Interface

```
ER ercd = tkse_loc_rec(W fd, W mode);
```

Parameter

| | | |
|---|------|---|
| W | fd | file descriptor |
| W | mode | lock mode (F_UNLOCK F_LOCK F_TSLOCK F_CKLOCK) F_LOCK lock setting (waiting) F_UNLOCK unlock F_TSLOCK lock setting (no waiting) F_CKLOCK check lock state |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------------|---|
| E_OK | Normal completion |
| E_ENDR | Current record is an end record |
| E_FD | File descriptor does not exist |
| E_IO | Input/output error occurred |
| E_LIMIT | The maximum number of records that can be locked simultaneously exceeded |
| E_LOCK | Current record is locked from others. Current record is already locked from others (when F_TSLOCK/F_CKLOCK is specified) It has already been locked from another file descriptor of its own process (when F_LOCK is specified) Lock cannot be released since it is locked by another file descriptor (when F_UNLOCK is specified) |
| E_DISWAI | Since the message handler has been started, WAIT processing was interrupted (when F_UNLOCK is specified) |
| E_PAR | Parameter is invalid (mode is invalid) |
| E_SYSTEMEM | Memory area of the system is insufficient |

Description

Locks the current record of the opened file.

When "F_LOCK" (lock (waiting)) is set and locked by other process, waits until it will be unlocked (waiting shall be in the priority order of process while the waiting with same priority shall be in the order of entering into WAIT state). Normal completion shall be executed without any processing when locked by the same file descriptor of invoking process. An error occurs when locked by the other file descriptor of invoking process.

In the case "F_UNLOCK" (unlock) is set, normal completion shall be executed without any processing when the record is unlocked. Unlock is enabled only when locked by the same file descriptor of this process, otherwise an error occurs.

In the case "F_TSLOCK" (lock (no waiting)) is set, an error occurs when the record was locked by other process or other file descriptor.

In the case "F_CKLOCK" (check lock state) is set, when the record was locked by other process or other file descriptor, an error occurs, and otherwise normal completion shall be executed without any processing.

The file descriptor other than the locked file descriptor is prohibited from reading, writing, resizing, modifying and deleting the locked record.

The lock set by the opened process is released at file close.

Check File Access Privileges

tkse_chk_fil

C Language Interface

```
ER ercd = tkse_chk_fil(LINK *lnk, W mode, TC *pwd);
```

Parameter

| | | |
|------|------|---|
| LINK | *lnk | target file |
| W | mode | check mode ([F_READ] [F_WRITE] [F_EXCUTE]) [F_EXIST] F_READ check access privileges to read (R) F_WRITE check access privileges to write (W) F_EXCUTE check access privileges to execute/search (E) F_EXIST check existence of file |
| TC | *pwd | password (valid only when "F_READ" or "F_WRITE" is specified) NULL no password specification |

Return Parameter

| | | |
|----|------|--|
| ER | ercd | < 0 error code = 0 normal completion (when non-"F_EXIST" is specified) >= 0 normal completion (file access information: when "F_EXIST" is specified) |
|----|------|--|

Error Code

| | |
|----------|--|
| E_FACV | No file (lnk) access right (when F_EXIST is specified) |
| E_MACV | Address (lnk) access is not permitted. |
| E_IO | Input/output error occurred |
| E_NOEXS | File (lnk) does not exist |
| E_NOFS | File system where file (lnk) belongs is not connected. |
| E_PAR | Parameter is invalid (mode is invalid) |
| E_PWD | The password of file (lnk) does not match (when F_READ/F_WRITE is specified) |
| E_RDONLY | Non-writable attribute of the file (when F_WRITE is specified) is set or file system that it belongs to is write-protected |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Checks whether the access specified by specified file is enabled or not.

An error occurs when the access specified in combination with "F_READ", "F_WRITE", and "F_EXCUTE" is disabled.

Since password function is currently unsupported, "pwd" is set to NULL.

In the case of "F_EXIST" specification, an error occurs when the file does not exist; when the file exists, the following access information is returned as return value:

0....0 BAPO SRWE

B: application attribute 2 (1: ON, 0: OFF)

A: application attribute 1 (1: ON, 0: OFF)

P: permanent attribute (1: ON, 0: OFF)

O: read-only attribute (1: ON, 0: OFF)

S: with or without password (1: with password, 0: without password)

R: access privileges to read (R) (1: with password, 0: without password)

W: access privileges to write (W) (1: with password, 0: without password)

E: access privileges to execute/search (E) (1: with password, 0: without password)

Change File Access Mode

tkse_chg_fmd

C Language Interface

```
ER ercd = tkse_chg_fmd(LINK *lnk, A_MODE *mode);
```

Parameter

| | | |
|--------|-------|---------------------------|
| LINK | *lnk | target file |
| A_MODE | *mode | access mode to be changed |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------------|--|
| E_OK | Normal completion |
| E_FACV | Not file (lnk) owner or level 0 user |
| E_MACV | Address (lnk, mode) access is not permitted |
| E_IO | Input/output error occurred |
| E_NOEXS | File (lnk) does not exist |
| E_NOFS | File system where file (lnk) belongs is not connected |
| E_PAR | Parameter is invalid. (contents of mode are invalid) |
| E_RDONLY | File (lnk) is unwritable or file system that it belongs to is unwritable |
| E_SYSTEMEM | Memory area of the system is insufficient |

Description

Changes access mode of the specified file.

“F_NOCHG” specification to the each of following data of access mode means that the item should not be changed.

- owner access mode (f_ownac)
- group access level (f_grpacc)
- public access level (f_pubacc)
- owner group number (f_grpno)

Regarding the change of access mode, when access level in the file system is zero, anyone can change the

access mode, however when access level is not zero, it can be changed by the process of the file owner only. When the access modes of the files that have been already opened are changed, the changes will not affect the files that have been already opened.

Change File Access Attribute

tkse_chg_fat

C Language Interface

```
ER ercd = tkse_chg_fat(LINK *Ink, W attr);
```

Parameter

| | | |
|------|------|--|
| LINK | *Ink | target file |
| W | attr | access attribute to be changed (F_SETONLY F_RSTRONLY F_SETPERM F_RSTPERM F_SETA1 F_RSTA1 F_SETA2 F_RSTA2) F_SETONLY set read-only attribute F_RSTRONLY reset read-only attribute F_SETPERM set permanent attribute F_RSTPERM reset permanent attribute F_SETA1 set application attribute 1 F_RSTA1 reset application attribute 1 F_SETA2 set application attribute 2 F_RSTA2 reset application attribute 2 |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | Normal completion |
| E_FACV | Not file (Ink) owner or level 0 user |
| E_MACV | Address (Ink) access is not permitted |
| E_IO | Input/output error occurred |
| E_NOEXS | File (Ink) does not exist |
| E_NOFS | File system where file (Ink) belongs is not connected. |
| E_PAR | Parameter is invalid (attr is invalid) |
| E_RDONLY | File (Ink) is unwritable or file system that it belongs to is unwritable |
| E_SYMEM | Memory area of the system is insufficient |

Description

Changes the access attribute to the specified file.

Regarding the change of access mode, when access level in the file system is zero, only the access mode can be changed by anyone. However when access level is not zero, it can be changed by the process of the file owner. When the access modes of the files that have been already opened are changed, the changes will not affect the files that have been already opened.

Change File name

tkse_chg_fnm

C Language Interface

```
ER ercd = tkse_chg_fnm(LINK *lnk, TC *name);
```

Parameter

| | | |
|------|-------|--|
| LINK | *lnk | target file |
| TC | *name | file name to be changed (valid to TNULL or the maximum number of file name characters) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | Normal completion |
| E_FACV | Not file (lnk) owner or level 0 user |
| E_MACV | Address (lnk, name) access is not permitted |
| E_IO | Input/output error occurred |
| E_NOEXS | File (lnk) does not exist |
| E_NOFS | File system where file (lnk) belongs is not connected. |
| E_FNAME | File name (name) is empty or invalid |
| E_PERM | File (lnk) cannot be deleted (deletion impossible attribute is set) |
| E_RDONLY | File is write-protected or file system that it belongs to is write-protected |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Changes the file name of the specified file.

Regarding the change of file name, when access level in the file system is zero, the file name can be changed by anyone. However when access level is not zero, it can be changed by the process of the file owner only.

The name of the file whose write-protected or unremovable attribute is set cannot be changed. When the specified file is a link file, both the name of destination file for reference and reference file held in the link file are changed.

Change File Date and Time

tkse_chg_ftm

C Language Interface

```
ER ercd = tkse_chg_ftm(LINK *lnk, F_TIME *times);
```

Parameter

| | | |
|--------|--------|----------------------------------|
| LINK | *lnk | target file |
| F_TIME | *times | date and time to be changed |
| | NULL | set to the current date and time |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------------|--|
| E_OK | Normal completion |
| E_FACV | Not file (lnk) owner or level 0 user |
| E_MACV | Address (lnk, times) access is not permitted |
| E_IO | Input/output error occurred |
| E_NOEXS | File (lnk) does not exist |
| E_NOFS | File system where the file (lnk) belongs is not connected. |
| E_RDONLY | File is write-protected or file system that it belongs to is write-protected |
| E_SYSTEMEM | Memory area of the system is insufficient |

Description

Changes the shelf life, the latest access time, and the last updated time of the specified file.

When each value of "F_TIME" is less than or equal to zero, the item is not changed.

When access level in the file system is zero, the file date and time can be changed. However when access level is not zero, it can be change by the process of the file owner only.

Get file information

tkse_fil_sts

C Language Interface

```
ER ercd = tkse_fil_sts(LINK *Ink, TC *name, F_STATE *stat, F_LOCATE *locat);
```

Parameter

| | | |
|----------|--------|---|
| LINK | *Ink | target file |
| TC | *name | storage area of file name (area for maximum file name + one character) NULL not stored |
| F_STATE | *stat | storage area of the file management information NULL not stored |
| F_LOCATE | *locat | storage area of the file location information NULL not stored |

Return Parameter

| | | |
|----|------|--|
| ER | ercd | < 0 error code >= 0 normal completion (reference count of the file) |
|----|------|--|

Error Code

| | |
|----------|---|
| E_MACV | Address (Ink,name,stat,locat) access is not permitted |
| E_IO | Input/output error occurred |
| E_NOEXS | File (Ink) does not exist |
| E_NOFS | File system where the file (Ink) belongs is not connected |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Retrieves the specified file information.

Get file information

tkse_ofl_sts

C Language Interface

```
ER ercd = tkse_ofl_sts(W fd, TC *name, F_STATE *stat, F_LOCATE *locat);
```

Parameter

| | | |
|----------|--------|---|
| W | fd | file descriptor |
| TC | *name | storage area of file name (area for maximum file name + one character) NULL not stored |
| F_STATE | *stat | storage area of the file management information NULL not stored |
| F_LOCATE | *locat | storage area of the file location information NULL not stored |

Return Parameter

| | | |
|----|------|--|
| ER | ercd | < 0 error code >= 0 normal completion (reference count of the file) |
|----|------|--|

Error Code

| | |
|----------|---|
| E_MACV | Address (name, stat, locat) access is not permitted |
| E_FD | File descriptor does not exist |
| E_IO | Input/output error occurred |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Retrieves the opened file information.

Get Link File Information

tkse_ink_sts

C Language Interface

```
ER ercd = tkse_ink_sts(LINK *Ink, F_LINK *stat);
```

Parameter

| | | |
|--------|-------|---|
| LINK | *Ink | target link file |
| F_LINK | *stat | storage area of the link file information |
| | NULL | not stored |

Return Parameter

| | | |
|----|------|---|
| ER | ercd | < 0 error code |
| | | >= 0 normal completion (reference count of the link file) |

Error Code

| | |
|----------|---|
| E_MACV | Address (Ink, stat) access is not permitted |
| E_IO | Input/output error occurred |
| E_NOEXS | File (Ink) does not exist |
| E_NOFS | File system where the file (Ink) belongs is not connected |
| E_NOLNK | Not a link file |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Retrieves the link file information of the specified link file.

When the specified file is not a link file, an error occurs.

Synchronize Link File

tkse_syn_Ink

C Language Interface

```
ER ercd = tkse_syn_Ink(LINK *Ink, W opt);
```

Parameter

| | | |
|------|------|---------------------------|
| LINK | *Ink | target link file |
| W | opt | synchronization attribute |
| | = 0 | check only |
| | != 0 | check and update |

Return Parameter

| | | |
|----|------|--|
| ER | ercd | < 0 error code |
| | | >= 0 normal completion (synchronization state) |

Error Code

| | |
|------------|--|
| E_MACV | Address (Ink) access is not permitted |
| E_IO | Input/output error occurred |
| E_NOEXS | File (Ink) does not exist |
| E_NOFS | File system where the file (Ink) or file at reference destination belongs is not connected |
| E_NOLNK | Not a link file |
| E_RDONLY | File is write-protected or file system that it belongs to is write-protected |
| E_SYSTEMEM | Memory area of the system is insufficient |

Description

Checks whether the file name, the generation time held by the specified link file are matched with the actual file name and generation time of the destination file for reference.

Only check shall be executed when “opt=0”. When “opt != 0” and the data does not match, updating shall be executed so that the information held by specified link file can be matched with the actual file name and generation time of the destination file for reference.

The return value indicates the following synchronization state:

| | |
|---------|---------------------|
| F_SYNC | matching |
| F_DNAME | different file name |

| | |
|---------|---|
| F_DDATE | different generation time |
| F_DBOTH | different in both file name and generation time |

When the specified file is not a link file, an error occurs.

Get default access mode

tkse_get_dfm

C Language Interface

```
ER ercd = tkse_get_dfm(DA_MODE *mode);
```

Parameter

```
DA_MODE *mode    storage area of default access mode
typedef struct {
    UH    f_ownacc;    /* owner access mode */
    UH    f_grpacc;    /* group access level */
    UH    f_pubacc;    /* public access level */
    H     f_grpno;     /* group number (0-4) */
    UH    f_gacc[N_GRP]; /* group access level */
} DA_MODE;
```

Return Parameter

```
ER    ercd    error code
```

Error Code

```
E_OK        Normal completion
E_MACV      Address (mode) access is not permitted
```

Description

Gets default access mode.

The default access mode is applied when the access mode is not specified at file generation.

While “f_gacc[4]” as a data just for reference indicates a group access level set to each user group, “f_grpacc” as an actual group access level is actually applied.

Set default access mode

tkse_set_dfm

C Language Interface

```
ER ercd = tkse_set_dfm(DA_MODE *mode);
```

Parameter

DA_MODE *mode default access mode to be set

Return Parameter

ER ercd error code

Error Code

| | |
|--------|---|
| E_OK | Normal completion |
| E_PAR | Parameter is invalid (content of mode is invalid) |
| E_MACV | Address (mode) access is not permitted |

Description

Sets default access mode.

The changed default access mode is valid for all processes.

Attach File System

tkse_att_fls

C Language Interface

```
ER ercd = tkse_att_fls(TC *dev, TC *name, LINK *lnk, UW mode);
```

Parameter

| | | |
|------|-------|---|
| TC | *dev | device name |
| TC | *name | connection name (valid to TNULL or maximum number of characters for connection name) |
| LINK | *lnk | storage area of the link to root file of the attached file system NULL not stored |
| UW | mode | connection mode (FS_SYNC FS_ASYN FS_RDONLY) FS_SYNC: synchronous write FS_ASYN: asynchronous write FS_RDONLY: read-only |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | Normal completion |
| E_FACV | No logical device(dev) access right (connection) |
| E_MACV | Address (dev, name, lnk) access is not permitted |
| E_BUSY | Logical device (dev) has already been opened or connected |
| E_OBJ | Connection name (name) already exists or a file system with same file system name has already been connected |
| E_FNAME | File name (name) is empty or invalid |
| E_IO | Input/output error occurred |
| E_LIMIT | The maximum number of file systems that can be connected simultaneously exceeded |
| E_NODEV | Access to device is not possible (dev) |
| E_NOEXS | Device (dev) is not registered or not a block type device |
| E_NOMDA | Media is not present in device (dev) |
| E_ILFMT | Not a standard file system format |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Connects up the file system in the specified device to the system by the specified connection name.

The connection name is used to specify the root file in the connected file system by absolute path name. It should not be the same as the connection names that have been already connected.

In the case "FS_SYNC" (synchronous write) is specified, writing into file is necessarily executed at the time of executing a write system call.

In the case "FS_ASYNC" (asynchronous write) is specified, writing into file is not necessarily executed at the time of executing a write system call.

In the case "FS_RDONLY" (read-only) is specified, every file write is prohibited.

An error occurs when reconnection of the connected file system is attempted or file system with the same file system name as the name of file system to which connecting is attempted is already connected.

To connect the file system, connection access privileges to the device is needed.

Detach File system

tkse_det_fls

C Language Interface

```
ER ercd = tkse_det_fls(TC *dev, W eject);
```

Parameter

| | | |
|----|-------|--|
| TC | *dev | device name |
| W | eject | specify to eject |
| | = 0 | not eject |
| | != 0 | eject (ignored, when device is unable to be ejected) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------------|--|
| E_OK | Normal completion |
| E_FACV | No logical device(dev) access right (connection) |
| E_MACV | Address (dev) access is not permitted |
| E_BUSY | File system is being used |
| E_IO | Input/output error occurred |
| E_NOEXS | Device (dev) is not registered or is not a block type device |
| E_NOFS | File system where the logical device (dev) belongs is not connected |
| E_NOMDA | Device media does not exist. |
| E_RDONLY | File cannot be written or file system that the file belongs to cannot be written |
| E_SYSTEMEM | Memory area of the system is insufficient |

Description

Detach the connected file system in the specified device from the system. At this time, when the content, etc. temporarily held in the memory exists, they shall be all written to the file system.

Detach is unavailable when a file in the file system targeted for detach is opened, or a process used as a working file exists.

To detach the file system, connection access privileges to the device is needed.

Synchronize File system

tkse_syn_fls

C Language Interface

```
ER ercd = tkse_syn_fls(void);
```

Parameter

none

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------------|--|
| E_OK | Normal completion |
| E_IO | Input/output error occurred |
| E_NOMDA | Device media does not exist. |
| E_RDONLY | File cannot be written or file system that the file belongs to cannot be written |
| E_SYSTEMEM | Memory area of the system is insufficient |

Description

The content, etc. temporarily held in the memory shall be all written to the file system, and the entire file system is updated to be existent for eliminating inconsistency. It shall be executed to all the connected file systems.

Get File system Management Information

tkse_flststs

C Language Interface

```
ER ercd = tkse_flststs(TC *dev, FS_STATE *buff);
```

Parameter

| | | |
|----------|-------|--|
| TC | *dev | device name |
| FS_STATE | *buff | storage area of the file system management information |

Return Parameter

| | | |
|----|------|---|
| ER | ercd | < 0 error code |
| | | = 0 normal completion (file system writable) |
| | | = 1 normal completion (file system read-only) |

Error Code

| | |
|----------|---|
| E_FACV | No logical device(dev) access right (connection) |
| E_MACV | Address (dev, buf) access is not permitted |
| E_BUSY | Logical device has already been opened. |
| E_IO | Input/output error occurred |
| E_NODEV | Access to device (dev) is not possible. |
| E_NOEXS | Device (dev) is not registered or not a block type device |
| E_NOMDA | Media is not present in device (dev) |
| E_ILFMT | Not a standard file system format |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Retrieves the management information of the connected file system in the specified device.

To retrieve the management information of the file system, connection access privileges to the device is needed.

Change File system Information

tkse_chg_fs

C Language Interface

```
ER ercd = tkse_chg_fs(TC *dev, TC *fs_name, TC *fs_locate);
```

Parameter

| | | |
|----|------------|--------------------------------------|
| TC | *dev | device name |
| | | An error occurs in the case of NULL. |
| TC | *fs_name | file system name to be changed |
| | NULL | No change |
| TC | *fs_locate | device location name to be changed |
| | NULL | No change |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | Normal completion |
| E_FACV | No logical device(dev) access right (connection, write) |
| E_MACV | Address (dev, fs_name, fs_locate) access is not permitted |
| E_BUSY | Logical device has already been opened. |
| E_OBJ | File system with specified file system name already exists (connected) |
| E_FNAME | File system name is empty or invalid |
| E_IO | Input/output error occurred |
| E_NODEV | Access to device (dev) is not possible |
| E_NOEXS | Device (dev) is not registered or not a block type device |
| E_NOMDA | Media is not present in device (dev) |
| E_RDONLY | File cannot be written or file system that the file belongs to cannot be written |
| E_ILFMT | Not a standard file system format |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Changes the names of both file system and the device location of the connected file system in the specified device.

To change file system information, connection access privileges and write access privileges to the device are needed.

Get Links Sequentially

tkse_get_nlk

C Language Interface

```
ER ercd = tkse_get_nlk(LINK *lnk);
```

Parameter

| | | |
|------|------|--|
| LINK | *lnk | link to the start file (input) |
| | | storage area of link to next file (output) |

Return Parameter

| | | |
|----|------|--|
| ER | ercd | < 0 error code |
| | | >= 0 normal completion (reference count of the retrieved file) |

Error Code

| | |
|----------|---|
| E_MACV | Address (lnk) access is not permitted |
| E_IO | Input/output error occurred |
| E_NOFS | File system where the file (lnk) belongs is not connected |
| E_NOEXS | File with a larger file ID than file (lnk) does not exist |
| E_SYSMEM | Memory area of the system is insufficient |

Description

Retrieve the link to a file with a minimum file ID among (out of) files with the file ID greater than the file ID of the specified start file.

The file ID of the start file can be that of nonexistent file in practice. All the attribute data to the link to the retrieved file is set to zero.

All links to the file (including reference count = 0 file) that exist in the file system can be taken out by sequentially acquiring the link from the route file of the file system (file ID = 0).

Get File system

tkse_lst_fls

C Language Interface

```
ER ercd = tkse_lst_fls(F_ATTACH *buff, W cnt);
```

Parameter

```
F_ATTACH *buff    storage area of the file system connection information (array)
                  typedef struct {
                      TC      a_name[L_CONNM]; /* connection name */
                      TC      dev[L_DEVNM];   /* logical device name */
                  } F_ATTACH;
```

```
W      cnt        > 0          Indicate the element number of "buff".
                  = F_GETDEV   Get device name.
                  = F_GETNAM   Get connection name.
```

Return Parameter

```
ER      ercd      < 0 error code
                  >= 0 normal completion (the number of the connected file systems)
                  = 1  normal completion (when "F_GETDEV" and "F_FETNAM" is specified)
```

Error Code

```
E_MACV    Address (buff) access is not permitted
E_NOFS    File system is not connected (When cnt= - 1, - 2)
E_PAR     Parameter is invalid (cnt=0,< - 2)
```

Description

Retrieve the connection name and the device name of the file systems that have been already connected.

When "cnt > 0", retrieve connection information of all the connected file systems to store in "buff".

When the number of the connected file systems is greater than the number of specified element (cnt), retrieve only the first pieces as many information as the number of element (cnt).

When "cnt" is set to "F_GETDEV", the device name corresponding to the connection name set to "buff->a_name[]" shall be stored into in "buff->dev[]".

When "cnt" is set to "F_GETNAM", the connection name corresponding to the device name set to "buff->dev[]"

shall be stored into in “buff->a_name[]”.

Map Record

tkse_map_rec

C Language Interface

```
ER ercd = tkse_map_rec(W fd, W offset, B **addr, W size, W mode);
```

Parameter

| | | |
|---|-----------|--|
| W | fd | file descriptor |
| W | offset | byte offset to start map |
| B | **addr | storage area of the mapped memory address |
| W | size | byte size to be mapped |
| W | mode | map mode |
| | | ([F_READ] [F_WRITE] [F_EXECUTE]) [F_COMMON F_SYSTEM] |
| | F_READ | map to read |
| | F_WRITE | map to write |
| | F_EXECUTE | map to execute |
| | F_COMMON | mapped to the shared memory space |
| | F_SYSTEM | mapped to the system memory space |

Return Parameter

| | | |
|----|------|--------------------------------|
| ER | ercd | < 0 error code |
| | | > 0 normal completion (map ID) |

Error Code

| | |
|---------|--|
| E_FD | File descriptor does not exist. |
| | Map mode contradicts the open mode |
| E_REC | Current record is a link record.. |
| E_MACV | Address (addr) access is not permitted |
| E_ENDR | Current record is an end record. |
| E_LOCK | Current record is locked from others |
| E_IO | Input/output error occurred |
| E_PAR | Parameter is invalid |
| E_NOSPT | Mapping no possible due to system restrictions |

Description

Map the “size” bytes from “offset” of the current record of the opened file to memory space. The content of the mapped record can be accessed as a memory.

When “F_COMMON” is specified, it is mapped to the shared memory space. In this case, the access from all processes is enabled. When “F_SYSTEM” is specified, it is mapped to the system memory space. In this case, accessed from all system processes is enabled. The general application process, even if it is the mapped process itself, cannot access from the public application process. Consequently, this setting shall not be used by an application process.

When neither “F_COMMON” nor “F_SYSTEM” is specified, it is mapped on the local memory space of the mapping process. In this case, access from the non-mapping process is not enabled.

The address to be mapped is determined by the system, and cannot be specified by the application.

The map mode specification shall be consistent with the opened mode. (E_FD)

- ”F_READ” open does not allow the mapping in “F_WRITE” mode.
- ”F_WRITE” open does not allow the mapping in “F_READ” and “F_EXECUTE” mode.

A link record cannot be mapped. (E_REC)

During mapping, the following manipulations are prohibited, and an E_BUSY error occurs.

- del_rec when the mapped record is targeted.
- wri_rec when the mapped record is targeted.
- trc_rec when the mapped record is targeted.
- xch_fil when the file including the mapped record is targeted.

Mapping is restricted in the following cases:

- file system in the removable media
- file system with logical block size less than 4KB

Unmap Record

tkse_ump_rec

C Language Interface

```
ER ercd = tkse_ump_rec(W fd, W mapid);
```

Parameter

| | | |
|---|-------|-----------------|
| W | fd | file descriptor |
| W | mapid | map ID |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|--------------------------------|
| E_OK | Normal completion |
| E_FD | File descriptor does not exist |
| E_NOEXS | Map ID does not exist |
| E_IO | Input/output error occurred |

Description

Release the map specified by map ID of the opened file. The map set by the opened process is released at file close.

Change File system Connection Mode

tkse_chg_fsm

C Language Interface

```
ER ercd = tkse_chg_fsm(TC *dev, UW mode);
```

Parameter

| | | |
|----|------|---|
| TC | *dev | logical device name |
| UW | mode | connection mode (same as the mode of att_fls) (FS_SYNC FS_ASYNC FS_RDONLY) FS_SYNC synchronization FS_ASYNC asynchronization FS_RDONLY read-only |

Return Parameter

| | | |
|----|------|--|
| ER | ercd | < 0 error code >= 0 normal completion (connection mode before change) |
|----|------|--|

Error Code

| | |
|--------|--|
| E_FACV | Logical device (dev) does not have a connection access right |
| E_MACV | Address (dev) access is not permitted |
| E_IO | Input/output error occurred |
| E_PAR | Parameter is invalid |
| E_NOFS | Logical device (dev) is not connected as a file system |

Description

Change the connection state of the device “dev” to the connection mode specified by “mode”. “dev” must be an device that has been already connected. The connection mode before change is returned as a return value.

To change the connection mode, connection access privileges to the device “dev” is required.

When writable attribute is set before the connection mode change, even if the attribute is changed to unwritable, the write mode record map mapped by “tkse_map_rec()” is valid and writing is enabled. In the case a file is “F_WRITE” or “F_UPDATE” open, when connection mode is changed to read-only, writing to the file by “tkse_wri_rec()”, etc. returns an “E_RDONLY” error code.

Synchronize on File Basis

tkse_syn_fil

C Language Interface

```
ER ercd = tkse_syn_fil(W fd);
```

Parameter

fd file descriptor

Return Parameter

ER ercd error code

Error Code

E_OK Normal completion
E_FD File descriptor does not exist

Description

Regarding the file in specified open, the content, etc. temporarily held in memory of specified open file shall be written to the disk. Also the content written on the record map (tkse_rec_map) shall be written to the disk

4.9 Filesystem Sharing Management

4.9.1 Filesystem Sharing Management Overview

The filesystem sharing management function of AMP TKSE is a function for sharing filesystems between AMP TKSE. A filesystem that has already been connected by other AMP TKSE becomes the source of sharing, and the filesystem can be operated as a filesystem of its own standard input/output management. In AMP TKSE, which is the source of sharing, the disk device must actually exist.

The filesystem sharing management function provides the functions of sharing filesystems and releasing shared filesystems only. Operations to shared filesystems are conducted by using the API of standard input/output management.

The filesystem type of shared filesystems is MOUNT_SFS (shared filesystem). Although operations can be conducted in the same way as other filesystems, it is not possible to detach the connection (tkse_detach). When the use of the shared filesystem ends, sharing is released.

When reading and writing data is conducted for files on the shared filesystem, the file buffer area to be specified must specify the shared memory between kernels. The targets are the standard input/output calls below.

- tkse_read
- tkse_write
- tkse_getdents
- tkse_getfsstat

4.9.2 System Call

Sharing of Filesystems

tkse_sharefs

C Language Interface

```
ER ercd = tkse_sharefs(ID kdmid, const char *sfsnm, const char *connm, int mode);
```

Parameter

| | | |
|-------------|-------|--|
| ID | kdmid | Kernel Domain ID |
| const char* | sfsnm | Connection name of a shared existing filesystem |
| const char* | connm | New connection name of a shared filesystem |
| int | mode | Connection mode [SF_RDONLY] SF_RDONLY 0x0001 Read only |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|--------|-----------------------------|
| E_OK | Normal completion |
| E_ID | kdmid is invalid |
| E_NOFS | Filesystem is not connected |
| E_OBJ | Already shared |

Description

Shares filesystems with the connection name specified by sfsnm as its own filesystem in the AMP TKSE of the kernel specified by kdmid. The connection name of the shared filesystem is connm, Its own AMP TKSE cannot be specified in kdmid.

sfsnm must be the connection name of the filesystem that has already been connected. This call shares the connected filesystem only, and a function to connect newly is not provided.

If NULL(0) is specified in common, the connection name of the shared filesystem becomes sfsnm.

If SF_RDONLY is specified in mode, the filesystem is shared by reading only.

In order to discontinue the use of a shared filesystem, release the sharing (tkse_removefs). It is impossible to detach the shared filesystem (tkse_detach). In addition, shared filesystems cannot be detached (tkse_detach) in AMP TKSE which is the source of sharing.

Filesystem Sharing Release

tkse_removefs

C Language Interface

```
ER ercd = tkse_removefs(const char *connm);
```

Parameter

| | | |
|-------------|-------|------------------------|
| const char* | connm | Device connection name |
|-------------|-------|------------------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|-------|--|
| E_OK | Normal completion |
| E_PAR | Not shared (sharing has already been released) |

Description

Releases the sharing of shared filesystems with the connection name specified by connm.

The shared filesystem cannot be used later.

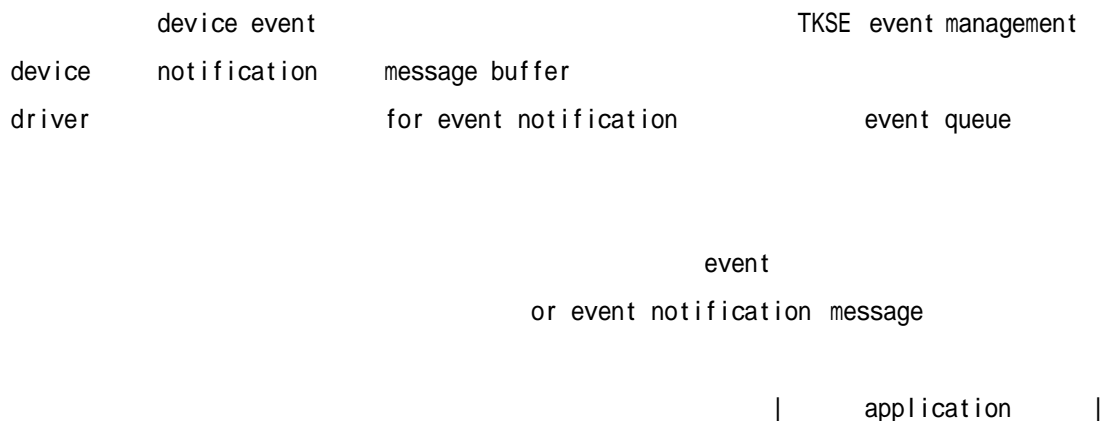
If a file which is being used (file that has been opened) exists in the specified shared filesystem, sharing cannot be released.

4.10 Event Management

4.10.1 Event Management Overview

The event management function of AMP TKSE uniformly treats notifications of phenomena from various devices as "events". The main purpose of event management function is to achieve interactive human interfaces. Therefore, keyboard and pointing devices are mainly assumed as the target devices of event management, but other devices can be supported.

The following figure describes the overview of event management scheme:



[Figure 12] Position of Event Management

The event notification from various devices is conveyed to the event management using the AMP T-Kernel device event notification function. The event management creates events based on event notification from devices to sequentially store in the event queue.

The interface between the event management and devices essentially shall be device event notifications only. The function to control devices directly from the event management is not provided. In addition, the event management does not depend upon specific devices.

An application sequentially fetches an event from the event queue to execute an action corresponding to it. The application also can get the notification of event occurrence by using the interprocess message function. Then, the notification ("tkse_brk_msg" call) of event occurrence in the interprocess message function is invoked from the event management when an event occurs.

At a certain point, it is premised on the rule that specific processes only fetch events by using the event management function. Applications should work according to the rule. The event management function itself does not provide a specific mechanism to ensure this rule.

The event management function targets the event queue of its own AMP TKSE only. Moreover, events stored in the event queue are only events from devices registered in the AMP T-Kernel of its own processor. Therefore, the

event management function cannot be used between kernels.

The API specification of the event management function is the same as in the T-Kernel Standard Extension Version 1.00 Specification.

4.10.2 Event Type

Events are distinguished by type number of 0-15 and, events of the following types are defined:

- type number 0 null event (EV_NULL):
the pseudo event indicating that a target event has not occurred.
- type number 1 button down event (EV_BUTDWN):
It occurs when the device button is pressed.
- type number 2 button up event (EV_BUTUP):
It occurs when the device button is released.
- type number 3 key down event (EV_KEYDWN):
It occurs when a key on the keyboard is pressed.
- type number 4 key up event (EV_KEYUP):
It occurs when a key on the keyboard is released.
- type number 5 auto repeat key event (EV_AUTKEY):
It periodically repeats when the target key of auto repeat is kept held down. The time (offset) taken until the first auto repeat key event from when the target key of auto repeat is pressed, and the recurrence interval (interval) can be set to any values.
- type number 6 device event (EV_DEVICE):
It is a generic event which occurs according to device's certain operation, and the content depends on the device. This event occurs when removable media such as a disk are installed.
- type number 7 extended device event (EV_EXDEV):
It is the device event (EV_DEVICE) appended by extended information.
Used for the device events which have extended information such as ucode (ubiquitous ID).
- type number 8 - 15 application event (EV_APPL1 - EV_APPL8):
Defined and used by applications and used as a communication function among applications.

A type mask corresponding to each event type is also defined and the target event type can be specified by this mask. The type mask is bit ready and the events of the type corresponding to "1" are targeted. However, a mask is

not defined for a null event in the nature.

[Table 2] Event Type Number and Event Type Mask

| Eventtype number type mask | | |
|----------------------------|----|--------------------|
| ----- | | |
| EV_NULL | 0 | ----- |
| EV_BUTDWN | 1 | EM_BUTDWN (0x0001) |
| EV_BUTUP | 2 | EM_BUTUP (0x0002) |
| EV_KEYDWN | 3 | EM_KEYDWN (0x0004) |
| EV_KEYUP | 4 | EM_KEYUP (0x0008) |
| EV_AUTKEY | 5 | EM_AUTKEY (0x0010) |
| EV_DEVICE | 6 | EM_DEVICE (0x0020) |
| EV_EXDEV | 7 | EM_EXDEV (0x0040) |
| EV_APPL1 | 8 | EM_APPL1 (0x0080) |
| : | : | : : : |
| EV_APPL8 | 15 | APPL8 (0x4000) |

Meanwhile, the following special masks are prepared as type masks:

| | |
|---------|--------|
| EM_NULL | 0x0000 |
| EM_ALL | 0x7fff |

4.10.3 Event Creation from Device Event Notifications

The event management creates the event (Chapter 9.5: EVENT Structure) from the event types (TDEvtTyp) for the device event notifications to be stored in the event queue. There are four event types, and each of them determines an event type as follows:

(1) Basic Event (TDEvtTyp = 0x0001 - 0x002F)

(A) Basic events other than keyboard / pointing device:

defined events:

| | | |
|----------------|------|------------------------------|
| TDE_unknown | 0x00 | undefined |
| TDE_MOUNT | 0x01 | media insert |
| TDE_EJECT | 0x02 | media eject |
| TDE_ILLMOUNT | 0x03 | illegal media insert |
| TDE_ILLEJECT | 0x04 | illegal media eject |
| TDE_REMOUNT | 0x05 | media reinsert |
| TDE_CARDBATLOW | 0x06 | card battery remaining alarm |

| | | |
|-----------------|------|----------------------|
| TDE_CARDBATFAIL | 0x07 | card battery failure |
| TDE_REQEJECT | 0x08 | media eject request |

→ These events are stored in the event queue as device event (EV_DEVICE).

Also, basic events other than the following keyboard and pointing device are device events (EV_DEVICE):

(B) Basic events such as keyboard / pointing device:

defined events:

| | | |
|-------------|------|------------------------|
| TDE_PDBUT | 0x11 | PD button state change |
| TDE_PDMOVE | 0x12 | PD position move |
| TDE_PDSTATE | 0x13 | PD state change |
| TDE_PDEXT | 0x14 | PD extended event |
| TDE_KEYDOWN | 0x21 | key down |
| TDE_KEYUP | 0x22 | key up |
| TDE_KEYMETA | 0x23 | meta key state change |

These events are stored in the event queue as button down (EV_BUTDWN), button up (EV_BUTUP), key down (EV_KEYDWN), and key up (EV_KEYUP).

(2) System Event (TDEvtTyp = 0x0030 - 0x007F)

defined events

| | | |
|---------------|------|---------------------------|
| TDE_POWEROFF | 0x31 | power switch off |
| TDE_POWERLOW | 0x32 | power remaining alarm |
| TDE_POWERFAIL | 0x33 | power failure |
| TDE_POWERUSUS | 0x34 | auto power suspend |
| TDE_POWERUPTM | 0x35 | time update |
| TDE_CKPWON | 0x41 | autopower on notification |

- The system events are not stored in the event queue.

(3) Event with Extended Information (TDEvtTyp = 0x0080 - 0x00FF)

defined events

nothing special.

- Events with extended information are stored in the event queue as an extended device event (EV_EXDEV).

(4) User-defined Event (TDEvtTyp = 0x0100 - 0xFFFF)

defined events

→ nothing special.

- The user-defined events are not stored in the event queue. Also, it is not used in event management as a general rule.

4.10.4 Priority of Event Queue and Event

The event queue is the queue exclusively prepared in the system for storing events, and the events are stored in the order they occur. If there is no space in the queue, newly occurring events, namely the newest event is not put in the queue to be discarded.

The events to be stored in the event queue are restricted by system event masks. That is, only the events of the type corresponding to the bit of system event mask "1" are put in the event queue. The events of the type corresponding to the bit of system event mask "0" are ignored by the entire system to be discarded.

At system startup, the event queue is blank while the system event mask is zero. And because the event management function does not work practically, the system event mask necessarily must be set to an appropriate value.

The following priorities are applied to the events according to each event type, and the event with the higher priority is fetched from the event queue. The events with same priority are fetched in the order they occur.

(1: highest priority - 6: lowest priority)

1. EV_APPL1-4
2. EV_BUTDWN, EV_BUTUP, EV_KEYDWN, EV_KEYUP
3. EV_AUTKEY
4. EV_DEVICE, EV_EXDEV
5. EV_APPL5-8
6. EV_NULL

The null event (EV_NULL) and the auto repeat event (EV_AUTKEY) are not actually put in the event queue, they are automatically created when condition is realized at an event fetch request.

4.10.5 Keyboard Events

(1) Regular Key and Meta Key

The keys are classified into regular keys which generate events when pressed or released and a meta key which does not generate events. The meta keys are keys which are used in combination with other regular keys, such as the shift key.

The following events are generated by pressing or releasing regular keys:

- Key down event (EV_KEYDOWN): generated when a keyboard key is pressed.
- Key up event (EV_KEYUP): generated when a keyboard key is released.
- an auto repeat event (EV_AUTKEY): generated periodically when the target key is kept held down.

(2) Auto Repeat Key

Auto repeat key events (EV_AUTKEY) generated when the target auto repeat key is kept held down.

When one key is held down and while holding down another key, only the auto repeat key event of the last pressed key generated.

The key targeted for auto repeat can be set to any keys except meta keys which generate no event key, and every key except meta keys can be the target of auto repeat at system startup.

System calls for setting/fetching the time (offset time) taken to the first occurrence from pressing and the subsequent recurrence interval (interval time) are provided. This time is in milliseconds. However, the recurrence interval of auto repeat key events depends on the implementation. The offset time and the interval time are rounded to the unit of event occurrence time.

4.10.6 Character Code

At key events, the key top code designating a key's physical location, and the encoded character code are returned.

The key top code is a fixed 8 bit code (0 - 255) according to key's physical location.

The character code is a code encoded by the state of key top code and the meta key. The encode is executed by drivers or hardware, etc. below event management, and has no concern in event management.

4.10.7 Pointing Device Event

The pointing device is used to select objects shown on the screen, and has absolute coordinate values as current position corresponding to the screen resolution. The absolute coordinate values are the coordinate values which have the origin (0, 0) in the upper-left corner of the screen and employ one pixel on the screen as a unit.

The following events are generated by pressing or releasing buttons of the pointing device:

- button down event (EV_BUTDWN): generated when device button is pressed.
- button up event (EV_BUTUP): generated when device button is released.

4.10.8 Designates the Operation Type of the Pointing Device

The pointing devices are classified into the following two types from their behaviors as a whole:

(1) Absolute operation type

Those tablet typed pointing devices such as a digital pen, etc. and their coordinate values are absolutely determined by the physical position of the pointing device.

(2) Relative operation type (differential type):

Those pointing devices with coordinate values that are relatively determined by the physical movement of the position of a pointing device such as mouse.

The calculation of absolute coordinate values in relative operation type is executed by drivers or hardware, etc. below event management and has no concern in event management.

4.10.9 Wheel Support

The wheel rotation of a wheel mouse is treated as an auto repeat key (EV_AUTKEY). However, the priority of an “EV_AUTKEY” event with the wheel is unlike those of the original “EV_AUTKEY” with key repeat, and is set to the lowest priority (higher than “EV_NULL”).

“EV_KEYDWN” and “EV_KEYUP” events by wheel rotation are not generated.

Wheel's events will automatically disappear from the event queue when either of the following conditions is realized:

- (1) An application did not fetch the wheel event within 300 ms after the wheel is turned.
- (2) An application fetched the events (except “EV_NULL”) other than the wheel event.

In addition, when the wheel was turned multiple times until an application fetches the event, the total amount is treated as one event. However, once the rotation direction is reversed, the previous rotation amount is discarded.

4.10.10 Event Structure

An Event is defined by the following structure:

```
typedef struct {
    W        type;        /* event type */
    UW       time;        /* event occurrence time */
    PNT      pos;         /* the position of pointing device when an event occurs */
    EVDATA   data;        /* event specific data */
    UW       stat;        /* meta key, button state */
    UB       exdat[16];   /* extended information */
} EVENT;
```

(1) type

The event type is indicated by a value in the range of 0-15.

(2) time

Indicate relative time in milliseconds, which shows the order and the interval of event occurrence and is meaningless as absolute time.

The event occurrence time is measured by an event timer. The event timer is implemented by using the system time management function in T-kernel. The event timer resolution depends on the implementation.

(3) pos

Indicate the coordinate values of the pointing device at an event occurrence in the value of absolute coordinates which have the origin (0, 0) in the upper-left corner of the screen. And it is the value of following "PNT" type.

```
typedef struct point {
    H        x;          /* horizontal coordinate value */
    H        y;          /* vertical coordinate value */
} PNT;
```

Meanwhile, the meaning of "pos" in the application events depends on the event definition.

(4) data

Indicate event-specific data and has event type-dependent content.

```
typedef union {
    struct { /* EV_KEYUP, EV_KEYDOWN, EV_AUTKEY */
        UH   keytop; /* key top code */
        TC   code;   /* character code */
    }
};
```

```

} key;
struct { /* EV_DEVICE, EV_EXDEV */
        H    kind; /* device event type */
        H    devno; /* device number */
} dev;
W    info; /* other data for event */
} EVDATA;

```

In the case of “EV_KEYDOWN”, “EV_KEYUP”, or “EV_AUTKEY”, keys are applied, with consisting of key top code, which indicates the key's physical position, and the encoded character code.

The content of “EV_AUTKEY” event generated by wheel rotation is as follows:

```

data.key.keytop :    0x8000 + rotation amount
                    rotation amount > 0 rotate the wheel forward
                    rotation amount < 0 rotate the wheel backward
data.key.code :    KC_SS_D          rotate the wheel forward
                   KC_SS_U          rotate the wheel backward

```

In the case of “EV_DEVICE”, “dev” is applied. And the type (kind) of device event and the device number (devno) which indicates the device generated by event are set. The event types are defined as follows:

```

kind = DE_unknown  0      -- undefined unknown
        DE_MOUNT   0x01   -- media insert
        DE_EJECT   0x02   -- media eject
        DE_ILLMOUNT 0x03   -- illegal media insert
        DE_ILLEJECT 0x04   -- illegal media eject
        DE_REMOUNT 0x05   -- media reinsert
        DE_BATLOW  0x06   -- battery remaining alarm
        DE_BATFAIL 0x07   -- battery failure
        DE_REQEJECT 0x08   -- media eject request
        0x09-     -- reserved
        0x7F

```

Also in the case of “EV_EXDEV”, “dev” is applied. And the type (kind) of device event and the device number (devno) which indicates the device generated by event are set. Usually, the event type of the device event notification is set only for the device event type (kind).

```

kind = XXXXXXXXXXXX 0x80- -- extended device event type (= device event type)
        0xFF

```

In the case of “EV_NULL”, “EV_BUTDOWN”, or “EV_BUTUP”, this “data” is not used, and “info” is always zero.

In the case of application event (EV_APPL1-8), an event definition-dependent content is set.

(5) stat

Indicate bit ready for the state of the key or the button which does not generate events alone, such as a meta key at an event occurrence. For each bit, "0" means released (OFF) state and "1" means pressed (ON) state. The meaning of each bit is defined as follows:

- bit 0 - 1 (2bit) PD basic button:
It indicates the state of main button and subbutton on the pointing device.
- bit 2 - 20 (19bit) meta key :
It indicates the state of meta keys on the keyboard.
The correspondence between each meta key and bit is undefined in the event management.
- bit 21 (1bit) PDtype :
It indicates the type of the pointing device.
- bit 22 - 23 (2bit) PD state :
It indicates the state of the pointing device.
- bit 24 - 31 (8bit) PD extended button:
It indicates the state of extended button on the pointing device.
The correspondence between each button and bit is not defined in the event management.

(6) exdat

Valid when the event type (type) is an extended device event (EV_EXDEV).

Used to store the extended information (16 bytes) mainly passed from the device event notification, such as ucode.

4.10.11 System Calls

Get Event

tkse_get_evt

C Language Interface

```
ER ercd = tkse_get_evt(W t_mask, EVENT* evt, W opt);
```

Parameter

| | | |
|--------|--------|--|
| W | t_mask | mask of target event type |
| EVENT* | evt | storage area of obtained event |
| W | opt | attribute of obtainment (CLR NOCLR) CLR eliminated from the event queue NOCLR not eliminated from the event queue |

Return Parameter

| | | | |
|----|------|------|---|
| ER | ercd | >= 0 | normal completion (obtained event type) |
| | | < 0 | error code |

Error Code

| | |
|----------|--|
| E_MACV | access to inaccessible address (evt) not allowed |
| E_IO | input/output error occurred (some device error occurred) |
| E_PAR | illegal parameter (t_mask <= 0, illegal "opt") |
| E_SYSMEM | insufficient system memory area |

Description

Fetch the event of the type specified by "t_mask" from the event queue to be stored in the area specified by "evt".

Without the event of specified type, "EV_NULL" is fetched.

The process when fetching an event from the event queue is specified by "opt".

Event Occurrence

tkse_put_evt

C Language Interface

```
ER ercd = tkse_put_evt(EVENT* evt, W opt);
```

Parameter

| | | |
|--------|-----|---|
| EVENT* | evt | event to occur |
| W | opt | occurrence attribute (EP_NONE EP_ALL ([EP_POS] [EP_STAT] [EP_TIME])) EP_NONE “time”, “pos”, and “stat” are set to the content of “evt” as-is. EP_ALL all of “time”, “pos”, and “stat” are set. EP_POS position of the current pointing device is set to “pos”. EP_STAT state of the current meta key/PD button is set to “stat”. EP_TIME value of the current event timer is set to “time”. |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | normal completion |
| E_MACV | access to inaccessible address (evt) not allowed |
| E_IO | input/output error occurred (some device error occurred) |
| E_PAR | illegal parameter (illegal event type, illegal “opt”) |
| E_SYSMEM | insufficient system memory area |

Description

Generate the event specified by “evt” and put in the event queue. An error occurs when the event queue is full or “EV_NULL” and “EV_AUTKEY” are specified. Also, events out of target in the system event mask are actually not generated and ignored.

The generated events are considered to be generated when this system call is executed, regardless of the “time” value, and are always put in the end of the event queue.

Clear Event

tkse_clr_evt

C Language Interface

```
ER ercd = tkse_clr_evt(W t_mask, W last_mask);
```

Parameter

| | | |
|---|-----------|---|
| W | t_mask | event type mask targeted to clear |
| | EM_ALL | all event types |
| W | last_mask | last event type mask to clear |
| | EM_ALL | clear only one event |
| | EM_NULL | target is to the end of the event queue |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|-------|--|
| E_OK | normal completion |
| E_PAR | illegal parameter ("t_mask <= 0", "last_mask < 0") |

Description

Clear generated events.

Out of the events in the event queue, events of the type specified by "t_mask" shall be cleared to the right before of the events of the type specified by the "last_mask". The events of the type of specified by the "last_mask" are not cleared. However, when "last_mask" = "EM_ALL", it means that only one event of the "last_mask" type is specifically cleared. Also, when "last_mask" = "EM_NULL", the target is set to the end of the event queue.

Examples to combine "t_mask" and the "last_mask" are shown as follows:

[Table 3] Combination of t_mask and last_mask

| t_mask | last_mask | behavior |
|--------|-----------|--|
| EM_ALL | EM_NULL | clear all events |
| -- | EM_ALL | clear only one event specified by "t_mask" |
| EM_ALL | EM_ALL | clear only the one start event |

Get Event Timer Value

tkse_get_etm

C Language Interface

```
RR ercd = tkse_get_etm(UW* time);
```

Parameter

| | | |
|----|-------|-----------------------------------|
| UW | *time | storage area of event timer value |
|----|-------|-----------------------------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|--------|---|
| E_OK | normal completion |
| E_MACV | access to inaccessible address (time) not allowed |

Description

Fetch the current value of the event timer.

The event timer is a relative time in milliseconds, but its actual resolution depends on the implementation.

Change System Event Mask

tkse_chg_emk

C Language Interface

```
ER o_mask = tkse_chg_emk(W mask);
```

Parameter

| | | |
|---|------|---|
| W | mask | system event mask to set |
| | < 0 | not change (obtain current system event mask) |

Return Parameter

| | | |
|----|--------|----------------------------|
| ER | o_mask | original system event mask |
|----|--------|----------------------------|

Description

Change the system event mask to the value specified by the “mask”, and return the value of the original system event mask as a return value.

When “mask < 0”, the value of current system event mask shall be returned as a return value without any changes.

Request Event Message

tkse_req_evt

C Language Interface

```
ER ercd = tkse_req_evt(W t_mask);
```

Parameter

W t_mask target event type mask

Return Parameter

ER ercd error code

Error Code

The messages are as follows:

| | |
|---------|--|
| E_OK | normal completion |
| E_PAR | illegal parameter |
| E_LIMIT | the system limit is exceeded by the number of registration of event message requests |

Description

When an event of the type specified by the “t_mask” is generated, sending of the event as a message to this process is requested. However, “EV_NULL” and “EV_AUTKEY” cannot be sent as messages. In addition, the event is not eliminated from the event queue even if it is sent as message.

The event message request is cleared by specifying “EM_NULL” to the “t_mask”. Also, it shall be automatically cleared after the process is completed.

```
struct {
    W        msg_type;     /* message type = MS_SYS5 */
    W        msg_size;     /* message size */
    EVENT    evt;         /* event */
    VB        info[];      /* additional information */
}
```

“msg_type” shall be fixed to “MS_SYS5”.

The generated events are stored in “evt”. Meanwhile, additional information may be stored in info according to event types. Therefore, “msg_size” is at least equivalent to “sizeof”(EVENT) and its size increases by the size of additional information. The size varies according to the content of additional information.

Additional information is appended in the case of device event (EV_DEVICE). Additional information is the content of the event notification itself from the device driver.

The sending process ID when receiving messages from “tkse_rcv_msg()” is the process ID when invoking “tkse_put_evt()”. In the case the event was generated by “tkse_put_evt()”. Otherwise, ID = 1 (initial process).

When sending is unsuccessful because the process message queue is full, the event message is simply discarded. The maximum number of processes to simultaneously execute the requests of process event message is limited by the system.

Get Elapsed Time from the Last Event Occurrence

tkse_las_evt

C Language Interface

```
ER ercd = tkse_las_evt(W t_mask);
```

Parameter

W t_mask target event type mask

Return Parameter

| | | | |
|----|------|------|---|
| ER | ercd | >= 0 | normal completion (elapsed time from the last event occurrence) |
| | | < 0 | error code |

Error Code

E_PAR illegal parameter

Description

Out of events of the type specified by “t_mask”, elapsed time shall be returned in milliseconds from the last generated event to the current time.

When “EM_BUTDWN” or “EM_BUTUP” is specified to “t_mask”, the pointer movement and the menu button operation are treated as event occurrences well. Also when “EM_KEYDWN” or “EM_KEYUP” is specified, the change of the meta key state is treated as an event occurrence.

The current time is set to the last occurrence time of events of all types by specifying “EM_NULL” to “t_mask”.

Set Auto Repeat Target Key

tkse_set_krm

C Language Interface

```
ER ercd = tkse_set_krm(KeyMap keymap);
```

Parameter

KeyMap keymap keymap targetd for auto repeat

Return Parameter

ER ercd error code

Error Code

E_OK normal completion
E_MACV access to inaccessible address (keymap) not allowed

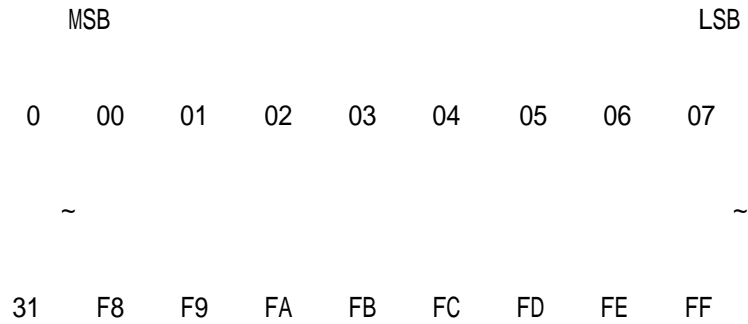
Description

Set the key targetd for auto repeat to the value specified by the “keymap”.

The “keymap” is an array which associates a single key (key top code) with one bit, and is defined as follows:

```
typedef UB KeyMap[KEYMAX/8];
```

“KEYMAX” is a maximum value of key top codes. The “keymap” structure is as follows:



number is key top code in hexadecimal

[Figure 13] KeyMap structure

The key corresponding to the bit of the specified keymap "1" is set to the target of the auto repeat, the key corresponding to the bit of keymap "0" is not set to the target of the auto repeat.

Meta keys which do not generate events are ignored even if they are set to the auto repeat target.

Get Auto Repeat Target Key

tkse_get_krm

C Language Interface

```
ER ercd = tkse_get_krm(KeyMap keymap);
```

Parameter

KeyMap keymap storage area of key map targeted for auto repeat

Return Parameter

ER ercd error code

Error Code

E_OK normal completion
E_MACV access to inaccessible address (keymap) not allowed

Description

Fetch the key targeted for auto repeat to be stored in the area specified by the “keymap”.

The key corresponding to the bit of the fetched keymap "1" is the target of the auto repeat while the key corresponding to the bit of keymap "0" is not the target of the auto repeat.

Set Auto Repeat Interval

tkse_set_krp

C Language Interface

```
ER ercd = tkse_set_krp(W offset, W interval);
```

Parameter

| | | |
|---|----------|---|
| W | offset | time taken to the occurrence of first auto repeat (in milliseconds) |
| W | interval | auto repeat recurrence interval (in milliseconds) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|-------|--|
| E_OK | normal completion |
| E_PAR | illegal parameter (offset<=0, interval<=0) |

Description

The time and interval until the occurrence of auto repeat key event (EV_AUTKEY) shall be set to the value specified by “offset” and “interval”.

The time is set in milliseconds, however its actual resolution depends on the implementation.

Get Auto Repeat Interval

tkse_get_krp

C Language Interface

```
ER ercd = tkse_get_krp(W* offset, W* interval);
```

Parameter

| | | |
|---|-----------|--|
| W | *offset | storage area of the elapsed time until the first occurrence of auto repeat (in milliseconds) |
| W | *interval | storage of the interval of the occurrence of auto repeat (in milliseconds) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|--------|---|
| E_OK | normal completion |
| E_MACV | access to inaccessible address ("offset", "interval") not allowed |

Description

Fetch the time and interval until the occurrence of auto repeat event (EV_AUTKEY), to be stored in the area specified by the "offset" and "interval".

Get PD Position

tkse_get_pdp

C Language Interface

```
ER ercd = tkse_get_pdp(PNT* pos);
```

Parameter

| | | |
|------|-----|-----------------------------|
| PNT* | pos | storage area of PD position |
|------|-----|-----------------------------|

```
typedef struct point {
    H    x;    /* horizontal coordinate value */
    H    y;    /* vertical coordinate value */
} PNT;
```

Return Parameter

| | | | |
|----|------|------|--|
| ER | ercd | >= 0 | type of occurred event (normal completion) |
| | | < 0 | error code |

Error Code

| | |
|--------|--|
| E_MACV | access to inaccessible address (pos) not allowed |
|--------|--|

Description

The position of the current pointing device is fetched in absolute coordinate values. The content of the event queue shall be unchanged.

The type of simultaneously occurring event shall be returned. "EV_NULL" shall be returned when no event occurs.

4.11 Device Management Function

4.11.1 Device Management Function Overview

The device management function of AMP TKSE provides a function to operate devices registered in the system by using the device management function of AMP T-Kernel.

AMP TKSE does not have the function to register devices.

The device management function targets devices registered in the AMP T-Kernel of its own processor only. It cannot operate devices of other AMP T-Kernel.

The API specification of the device management function is the same as the T-Kernel Standard Extension Version 1.00 Specification.

4.11.2 Basic Concepts

(1) Device Name (UB* type)

A device name is a string of up to 8 characters consisting of the following elements.

```
#define L_DEVNM 8 /* Device name length */
```

- Type:

Name indicating the device type

Characters a to z and A to Z can be used.

- Unit:

One letter indicating a physical device

Each unit is assigned a letter from a to z in order starting from a.

- Subunit:

One to three digits indicating a logical device

Each subunit is assigned a number from 0 to 254 in order starting from 0.

Device names take the format type + unit + subunit. Some devices may not have a unit or subunit, in which case the corresponding field is omitted.

A name consisting of type + unit is called a physical device name. A name consisting of type + unit + subunit may be called a logical device name to distinguish it from a physical device name. If there is no subunit, the physical device name and logical device name are identical. The term "device name" by itself means the logical device name.

A subunit generally refers to a partition on a hard disk, but can be used to mean other logical devices as well.

Examples:

| | |
|------|---------------------------|
| hda | Hard disk (entire disk) |
| hda0 | Hard disk (1st partition) |
| fda | Floppy disk |
| rsa | Serial port |
| kbpd | Keyboard/pointing device |

(2) Device ID (ID type)

By registering a device (device driver) with T-Kernel/SM, a device ID (> 0) is assigned to the device (physical device name). Device IDs are assigned to each physical device. The device ID of a logical device consists of the device ID assigned to the physical device to which is appended the subunit number + 1 (1 to 255).

The device ID assigned at device registration

| | |
|---------------|----------------------------------|
| devid | Physical device |
| devid + n + 1 | The nth subunit (logical device) |

Examples:

| | | |
|------|-----------|----------------------------|
| hda | devid | Entire hard disk |
| hda0 | devid + 1 | 1st partition of hard disk |
| hda1 | devid + 2 | 2nd partition of hard disk |

(3) Device Attribute (ATR type)

Device attributes are defined as follows, in order to classify devices by their properties.

```
IIII IIII IIII IIII PRxx xxxxx KKKK KKKK
```

The high 16 bits are device-dependent attributes defined for each device. The low 16 bits are standard attributes defined as follows.

```
#define TD_PROTECT          0x8000    /* P: write protection */
#define TD_REMOVABLE        0x4000    /* R: removable media */
#define TD_DEVKIND          0x00ff    /* K: device/media kind */
#define TD_DEVTYPE          0x00f0    /* device type */
```

```
/* device type */
```

```
#define TDK_UNDEF           0x0000    /* undefined/unknown */
#define TDK_DISK            0x0010    /* disk device */
```

```
/* disk kind */
```

```
#define TDK_DISK_UNDEF     0x0010    /* miscellaneous disk */
#define TDK_DISK_RAM       0x0011    /* RAM disk */
```

```

#define   TDK_DISK_ROM      0x0012    /* ROM disk */
#define   TDK_DISK_FLA     0x0013    /* Flash ROM or other silicon disk*/
#define   TDK_DISK_FD      0x0014    /* floppy disk*/
#define   TDK_DISK_HD      0x0015    /* hard disk*/
#define   TDK_DISK_CDRROM  0x0016    /* CD-ROM*/

```

Currently no device types other than disks are defined. Other devices are assigned to undefined type (TDK UNDEF). Note that device types are defined for the sake of distinguishing devices from the standpoint of the user as necessary, such as when applications must change their processing based on the type of device or media. Devices for which no such distinctions are necessary do not have to have a device type assigned. See the individual device driver specifications regarding device-dependent attributes.

(4) Device Descriptor (ID type)

A device descriptor (> 0) is an identifier used for accessing a device, assigned by T-Kernel/SM when a device is opened.

A device descriptor belongs to a resource group. Operations using a device descriptor can be performed only by tasks belonging to the same resource group as the device descriptor.

(5) Request ID (ID type)

When an IO request is made to a device, a request ID (> 0) is assigned identifying the request. This ID can be used to wait for IO completion.

(6) Data Number (INT type)

Device data is specified by a data number. Data is classified into device-specific data and attribute data as follows.

- Device-specific data: Data number ≥ 0

As device-specific data, the data numbers are defined separately for each device.

Examples:

Disk Data number = physical block number

Serial port Data number = 0 only

- Attribute data: Data number < 0

Attribute data specifies driver or device state acquisition and setting modes, and special functions, etc.

4.11.3 System Calls

Open Device

tkse_opn_dev

C Language Interface

```
ID dd = tkse_opn_dev( UB *devnm, UINT omode );
```

Parameter

| | | |
|------|-----------|---|
| UB* | devnm | device name |
| UNIT | o_mode | open mode |
| | | (TD_READ TD_WRITE TD_UPDATE) [TD_EXCL TD_WEXCL TD_REXCL] |
| | TD_READ | open for reading |
| | TD_WRITE | open for writing |
| | TD_UPDATE | open for updating (reading/writing) |
| | TD_EXCL | exclusive |
| | TD_WEXCL | exclusive write |
| | TD_REXCL | exclusive read |

Return Parameter

| | | | |
|----|----|-----|---------------------------------------|
| ID | dd | > 0 | device descriptor (normal completion) |
| | | < 0 | error code |

Error Code

| | |
|---------|--|
| E_MACV | access to address (dev, error) not allowed |
| E_BUSY | the device (dev) is already opened exclusively. |
| E_OACV | the read or write processing to the device (dev) is not allowed. |
| E_NOEXS | device (dev) does not exist (not registered) |
| E_LIMIT | maximum number of open devices exceeded |
| Others | errors returned from device drivers |

Description

Opens the device specified by `dev` in the mode specified by `o_mode`, and returns a device descriptor if successful. The exclusive mode and exclusive write mode limit the opening of one device at the same time. When the process which opened a device exits, the device will be automatically closed.

Close Device

tkse_cls_dev

C Language Interface

```
ER ercd = tkse_cls_dev( ID dd, UINT option );
```

Parameter

| | | |
|------|----------|---|
| ID | dd | device descriptor |
| UINT | option | close option |
| | TD_EJECT | media eject (ignored for unejectable devices) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|--------|--|
| E_OK | normal completion |
| E_ID | no device descriptor present |
| Others | error codes returned from device drivers |

Description

Closes the device specified by dd.

Read Device Data (Asynchronous)

tkse_rea_dev

C Language Interface

```
ID reqid = tkse_rea_dev( ID dd, INT start, VP buf, INT size, TMO tmout );
```

Parameter

| | | |
|-----|-------|--|
| ID | dd | device descriptor |
| INT | start | start position for reading (≥ 0 : specific data, < 0 : attribute data) |
| VP | buf | storage area of read data |
| INT | size | read data size |
| TMO | tmout | timeout for request accept (millisecond) |

Return Parameter

| | | | |
|----|-------|-------|--------------------------------|
| ID | reqid | > 0 | request ID (normal completion) |
| | | < 0 | error code |

Error Code

| | |
|---------|---|
| E_MACV | not allowed to access address |
| E_ID | there exists no device descriptor, or it is D_WRITE opened. |
| E_OACV | the read processing to the device (dd) is not allowed. |
| E_NOMDA | Media is not present in device (dd) |
| E_LIMIT | maximum number of requests exceeded |
| E_TMOUT | timeout |
| E_ABORT | abort |
| Others | error codes returned from device drivers |

Description

Starts to read the data of the device specified by dd from the start position specified by start by the size specified by size to the area specified by buf.

Returns to its caller without waiting for completion of reading. It is necessary to maintain the buf area until reading is complete. When reading normally begins, the request ID is returned as the return value.

If size = 0, actual reading is not performed, but current readable data size is checked.

Read Device Data (Synchronous)

tkse_srea_dev

C Language Interface

```
ER ercd = tkse_srea_dev( ID dd, INT start, VP buf, INT size, INT *asize );
```

Parameter

| | | |
|------|-------|---|
| ID | dd | device descriptor |
| INT | start | start position for reading (>= 0: specific data, < 0: attribute data) |
| VP | buf | storage area of read data |
| INT | size | read data size |
| INT* | asize | read data size |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|---|
| E_MACV | not allowed to access address |
| E_ID | there exists no device descriptor, or it is D_WRITE opened. |
| E_OACV | the read processing to the device (dd) is not allowed. |
| E_NOMDA | no device (dd) media present |
| E_LIMIT | maximum number of requests exceeded |
| E_ABORT | abort |
| Others | error codes returned from device drivers |

Description

Read the data of the device specified by dd from the start position specified by start by the size specified by size to the area specified by buf.

When reading is finished, the control is returned to the caller, and the actual read size is set to (*asize).

If size = 0, actual reading is not performed, but current readable data size is returned to asize.

Write Data to Device (Asynchronous)

tkse_wri_dev

C Language Interface

```
ID reqid = tkse_wri_dev( ID dd, INT start, VP buf, INT size, TMO tmout );
```

Parameter

| | | |
|-----|-------|---|
| ID | dd | device descriptor |
| INT | start | start position for writing (>= 0: specific data, < 0: attribute data) |
| VP | buf | storage area of write data |
| INT | size | read data size |
| TMO | tmout | timeout for request accept (millisecond) |

Return Parameter

| | | | |
|----|-------|-----|--------------------------------|
| ID | reqid | > 0 | request ID (normal completion) |
| | | < 0 | error code |

Error Code

| | |
|----------|--|
| E_MACV | access to address (buf, a_size, error) not allowed |
| E_ID | there exists no device descriptor, or it is D_READ opened. |
| E_OACV | the write processing to the device (dev) is not allowed. |
| E_NOMDA | Media is not present in device (dd) |
| E_RDONLY | write-protected device (dev) |
| E_LIMIT | maximum number of requests exceeded |
| E_TMOUT | timeout |
| E_ABORT | abort |
| Others | error codes returned from device drivers |

Description

Starts to write data to the device specified by dd from the start position specified by start by the size specified by size from the area specified by buf.

Returns to its caller without waiting for completion of writing. It is necessary to maintain the area and content of buf until reading is complete. When reading normally begins, the request ID is returned as the return value.

If size = 0, actual writing is not performed, but current writable data size is checked.

Write Data to Device (Synchronous)

tkse_swri_dev

C Language Interface

```
ER ercd = tkse_swri_dev( ID dd, INT start, VP buf, INT size, INT *asize );
```

Parameter

| | | |
|-----|-------|---|
| ID | dd | device descriptor |
| INT | start | start position for writing (>= 0: specific data, < 0: attribute data) |
| VP | buf | storage area of write data |
| INT | size | read data size |
| INT | asize | read data size |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|----------|--|
| E_OK | normal completion |
| E_MACV | access to address (buf, a_size, error) not allowed |
| E_ID | there exists no device descriptor, or it is D_READ opened. |
| E_OACV | the write processing to the device (dev) is not allowed. |
| E_NOMDA | no device (dd) media present |
| E_RDONLY | write-protected device (dev) |
| E_LIMIT | maximum number of requests exceeded |
| E_ABORT | abort |
| Others | error codes returned from device drivers |

Description

Write data to the device specified by dd from the start position specified by start by the size specified by size from the area specified by buf. When the writing is finished, the control is returned to the caller, and the written size is set to *asize.

If size = 0, actual writing is not performed, but current writable data size is returned to asize.

Wait for Request Completion for Device

tkse_wai_dev

C Language Interface

```
ID reqid = tkse_wai_dev( ID dd, ID reqid, INT *asize, ER *ioer, TMO tmout );
```

Parameter

| | | |
|------|-------|-----------------------|
| ID | dd | device descriptor |
| ID | reqid | request ID |
| INT* | asize | read/write size |
| ER* | ioer | input/output error |
| TMO | tmout | timeout (millisecond) |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|---------|--|
| E_ID | illegal dd or reqid |
| E_OBJ | reqid request is waiting for other task's completion |
| E_NOEXS | no processing request (when reqid = 0) |
| E_TMOUT | timeout |
| E_ABORT | abort |
| Others | error codes returned from device drivers |

Description

Waits for reqid request's completion for the device specified by dd.

If reqid = 0, the completion of one of the requests for dd is waited.

Request to Suspend Device

tkse_sus_dev

C Language Interface

```
ER ercd = tkse_sus_dev(UINT mode);
```

Parameter

| | | |
|------|------------|-------------------------------------|
| UINT | mode | mode specification |
| | | D_EMRGUSUS D_SUSPEND [D_FORCE] |
| | | D_DISSUS D_ENASUS D_CHECK |
| | | D_NOTIFY [D_NOTSUS] [D_NOTRES] |
| | D_SUSPEND | suspend |
| | D_DISSUS | disable suspend |
| | D_ENASUS | enable suspend |
| | D_CHECK | check suspend prohibit count |
| | D_EMRGUSUS | emergency suspend |
| | D_FORCE | forced suspend specification |
| | D_NOTIFY | notification request |
| | D_NOTSUS | notification to suspend |
| | D_NOTRES | resume notification |

Return Parameter

| | | | |
|----|------|------|--|
| ER | ercd | >= 0 | normal completion (if D_CHECK, suspend prohibit request count) |
| | | < 0 | error code |

Error Code

| | |
|----------|---|
| E_BUSY | unable to suspend because of suspend inhibited state |
| E_PAR | illegal parameter |
| E_LIMIT | suspend prohibit request count limit exceeded |
| E_DISWAI | processing suspended because message handler is invoked |

Description

Conducts suspend related control of systems specified by mode specification "mode".

Retrieve Device Name

tkse_get_dev

C Language Interface

```
ID pdid = tkse_get_dev( ID devid, UB *devnm )
```

Parameter

| | | |
|-----|-------|-----------------------------|
| ID | devid | device ID |
| UB* | devnm | storage area of device name |

Return Parameter

| | | | |
|----|------|------|--|
| ID | pdid | >= 0 | normal completion (physical device ID) |
| | | < 0 | error code |

Error Code

| | |
|---------|-------------------------------------|
| E_MACV | access to address (dev) not allowed |
| E_NOEXS | device ID does not exist |

Description

Retrieves the device name of the device with device ID specified by devno, and stores it to the area specified by devnm.

Then, the device ID of the physical device to which the device belongs is returned as a return value.

The specified device ID is a device number fetched by a device event.

Retrieve Device Information

tkse_ref_dev

C Language Interface

```
ID  devid = tkse_ref_dev( UB *devnm, T_RDEV *rdev );
```

Parameter

| | | |
|---------|-------|---|
| UB* | devnm | target device name |
| T_RDEV* | rdev | storage area of device management information |

Return Parameter

| | | | |
|----|-------|------|-------------------------------|
| ID | devid | >= 0 | normal completion (device ID) |
| | | < 0 | error code |

device management information

```
typedef struct {
    ATR    devatr;    /* device attribute */
    INT    blkosz;   /* physical block size (-1: unknown) */
    INT    nsub;     /* number of subunits */
    INT    subno;    /* 0: physical device, 1 - nsub: subunit number + 1 */
} T_RDEV;
```

Error Code

| | |
|---------|-------------------------------|
| E_MACV | not allowed to access address |
| E_NOEXS | device does not exist |

Description

Retrieves information of the device specified by devnm, and stores it to the area specified by rdev.

Retrieve Device Information

tkse_oref_dev

C Language Interface

```
ID  devid = tkse_oref_dev( ID dd, T_RDEV *rdev );
```

Parameter

| | | |
|---------|------|---|
| ID | dd | target device descriptor |
| T_RDEV* | rdev | storage area of device management information |

Return Parameter

| | | | |
|----|-------|------|-------------------------------|
| ID | devid | >= 0 | normal completion (device ID) |
| | | < 0 | error code |

device management information

```
typedef struct {
    ATR    devatr;    /* device attribute */
    INT    blksize;  /* physical block size (-1: unknown) */
    INT    nsub;     /* number of subunits */
    INT    subno;    /* 0: physical device, 1 - nsub: subunit number +1 */
} T_RDEV;
```

Error Code

| | |
|---------|-------------------------------|
| E_MACV | not allowed to access address |
| E_NOEXS | device does not exist |

Description

Retrieves information of the device specified by dd, and stores it to the area specified by rdev.

Retrieve Registered Devices

tkse_lst_dev

C Language Interface

```
INT num = tkse_lst_dev( T_LDEV *ldev, INT start, INT ndev );
```

Parameter

| | | |
|--------|-------|---|
| T_LDEV | *ldev | storage area of registered device information (array) |
| INT | start | start number |
| INT | ndev | retrieved number |

Return Parameter

| | | | |
|-----|-----|------|---|
| INT | num | >= 0 | normal completion (remaining registered number) |
| | | < 0 | error code |

```
typedef struct {
    ATR    devatr;    /* device attribute */
    INT    blkosz;    /* specific data's block size (-1: unknown)
    INT    nsub;      /* number of subunits */
    UB     devnm[L_DEVNM]; /* physical device name */
} T_LDEV;
```

Error Code

| | |
|---------|-------------------------------------|
| E_MACV | access to address (dev) not allowed |
| E_NOEXS | start exceeds the registered number |
| E_PAR | illegal parameter (ndev < 0) |

Description

Retrieves registered device information and stores it to the area specified by ldev. Then, remaining device number is returned as a return value.

4.12 Time Management

4.12.1 Time Management Overview

The time management function of AMP TKSE provides such functions as to retrieve and set the system time as base time held within system, and to convert between system time and calendar date and time.

The system time is used to represent system's internal times such as the date and time to create, update or access files.

The system time is a 32 bit value represented by seconds since the date and time at 00:00:00 GMT (Greenwich Mean Time), Jan 1, 1985, and is defined as follows:

```
typedef W STIME;
```

As opposed to system time, the time of the region in which the machine actually exists is called local time. The time management function also holds the time difference between system time and local time, and provides functions to retrieve and set date and time based upon local time.

The relationship between system time and local time is defined as time compensation data as follows:

```
typedef struct {
    W      adjust;      /* time difference with system time (second) */
    W      dst_flg;     /* DST application type */
    W      dst_adj;     /* DST running-in time (minute) */
} TIMEZONE;
```

dst_flg indicates the application type of Daylight Saving Time (DST), and zero value indicates no application. The values other than zero indicate application. The value other than zero is meaningless in the time management function, only whether the data is zero or not is important.

dst_adj is a value in the range - (12 x 60) - + (12 x 60) which indicates DST running-in time (minute). The time management function does not determine whether to apply DST or not. At the start time of the period during which DST is actually applied, the system program is assumed to set an appropriate value to dst_adj, and at the end time of the period during which DST is applied, it is assumed to set zero to dst_adj.

The time compensation data allows you to define local time with the following expression:

```
local time (second)
    = system time (second) - adjust + (dst_flg ? (dst_adj x 60): 0)
```

The time management function also supports calendar date and time defined by the following structure, and provides a function to convert between this and system time:

```
typedef struct {
    W      d_year;      /* offset from 1985 (85-) */
    W      d_month;    /* month ( 1 - 12, 0) */
    W      d_day;      /* day ( 1 - 31 ) */
    W      d_hour;     /* hour ( 0 - 23 ) */
    W      d_min;      /* minute ( 0 - 59 ) */
    W      d_sec;      /* second ( 0 - 59 ) */
    W      d_week;     /* week ( 1 - 54 ) */
    W      d_wday;     /* a day of the week (0 - 6, 0: Sunday) */
    W      d_days;     /* day ( 1 - 366 ) */
} DATE_TIM;
```

d_week represents a week number when week 1 starts at the week of Jan 1 of the year, and d_days represents a day number when day 1 starts on Jan 1 of the year. In addition, d_month = 0 is used to hold a special meaning.

The API specification of the time management function is the same as the T-Kernel Standard Extension Version 1.00 Specification.

4.12.2 System Calls

Refer System Time

tkse_get_tim

C Language Interface

```
ER ercd = tkse_get_tim(SYSTIM *pk_tim);
```

Parameter

| | | |
|-------------------|--------|---|
| SYSTIM* | pk_tim | packet address which returns current time |
| content of pk_tim | | |
| SYSTIM | system | current time for system setting |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|-------|------------------------------------|
| E_OK | normal completion |
| E_PAR | illegal parameter (illegal pk_tim) |

Description

Reads current value of the system time and returns it in the pk_tim.

Set System Time

tkse_set_tim

C Language Interface

```
ER ercd = tkse_set_tim(SYSTIM *pk_tim);
```

Parameter

SYSTIM* pk_tim packet address which indicates current time

content of pk_tim

SYSTIM systim current time for system setting

Return Parameter

ER ercd error code

Error Code

E_OK normal completion

E_PAR illegal parameter (illegal pk_tim)

Description

Sets the value denoted by pk_tim to system time value.

Refer System Uptime

tkse_get_otm

C Language Interface

```
ER ercd = tkse_get_otm(SYSTIM *pk_tim);
```

Parameter

| | | |
|---------------|-------------------|-------------------------------------|
| SYSTIM* | pk_tim | packet address which returns uptime |
| | content of pk_tim | |
| SYSTIM opetim | | current time for system setting |

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|-------|------------------------------------|
| E_OK | normal completion |
| E_PAR | illegal parameter (illegal pk_tim) |

Description

Retrieves the system uptime.

The system uptime is different from system time, and it represents simply increasing uptime from system startup. Not affected by the time setting using `tkse_set_tim`. The system uptime should have the same precision as system time.

4.12.3 Library Calls

Conversion functions for system time, calendar date, local time, and Greenwich mean time (GMT).

The valid value for total number of seconds is in the range of $0x00000000-24*60*60 - 0x7ffffff+24*60*60$ (including a day before or after system time for time compensation by TIMEZONE). The valid years are 1985-2053. For functions which take calendar time as a parameter, the operation is not ensured when illegal calendar time is specified.

Convert Calendar Date to total number of seconds**DATEtoTIME****C Language Interface**

```
void DATEtoTIME(STIME *time, DATE_TIM *date);
```

Parameter

| | | |
|----------|-------|---|
| STIME | *time | storage area of total number of seconds |
| DATE_TIM | *date | calendar time |

Return Parameter

none

Description

Convert the calendar date specified by “date” to total number of seconds starting from 00:00:00 GMT, Jan 1, 1985 to be stored in “time”.

Convert consecutive seconds the total number of seconds to Calendar Date TIMEtoDATE

C Language Interface

```
void TIMEtoDATE(DATE_TIM *date, STIME time);
```

Parameter

| | | |
|----------|-------|-------------------------------|
| DATE_TIM | *date | storage area of calendar time |
| STIME | time | total number of seconds |

Return Parameter

none

Description

Convert the total number of seconds specified by “time” starting from 00:00:00 GMT, Jan 1, 1985 to calendar time to be stored in “date”.

local time compensation**GMTtoLT****C Language Interface**

```
STIME ltim = GMTtoLT(STIME time, TIMEZONE *tz);
```

Parameter

| | | |
|----------|------|---|
| STIME | time | storage area of total number of seconds |
| TIMEZONE | *tz | time compensation |

Return Parameter

| | | |
|-------|------|------------|
| STIME | ltim | local time |
|-------|------|------------|

Description

Apply time compensation specified by “tz” to system (GMT) time specified by “time”, and return the time after the conversion to local time.

standard time compensation**LTtoGMT****C Language Interface**

```
STIME gtim = LTtoGMT(STIME time, TIMEZONE *tz);
```

Parameter

| | | |
|----------|------|-------------------|
| STIME | time | |
| TIMEZONE | *tz | time compensation |

Return Parameter

| | | |
|-------|------|-------------------|
| STIME | gtim | system (GMT) time |
|-------|------|-------------------|

Description

Apply time compensation specified by “tz” to local time specified by “time”, and return the time after the conversion to local time to system (GMT) time.

4.13 System Management Function

4.13.1 System Management Function Overview

The system management function of AMP TKSE provides a function to load and unload system programs, and a function to acquire system information.

(1) Load and Unload of System Programs

System programs are AMP T-Kernel based programs arranged in system memory. System programs operate at the same protection level (level 0) as AMP T-Kernel, and the API of AMP T-Kernel can be executed and system memory can be accessed.

System programs are mainly used for the registration of device drivers and subsystems.

(2) Acquisition of System Information

AMP TKSE version information can be acquired as system information.

The API specification of the system management function is the same as the T-Kernel Standard Extension Version 1.00 Specification.

4.13.2 System Calls

Load System Programs

tkse_lod_spg

C Language Interface

```
ER ercd = tkse_lod_spg(T_LSPG *pk_sysprg, TC *arg, VW info[N_SPG_INFO]);
```

Parameter

| | | |
|--------|------------------|---|
| T_LSPG | *pk_sysprg | system program information |
| TC* | arg | string passed as arguments during loading system programs |
| VW | info[N_SPG_INFO] | storage area of loading information (N_SPG_INFO=2) |

```
typedef struct {
    ATR    spgatr    system program attribute
    VP     spghdr    handle to system program to load

    /* other implementation-dependent information */
} T_LSPG;
```

spgatr indicates process attribute and specifies the following:

```
spgatr := (TMA_SEIO || TMA_LINK || TMA_PTR)
```

TMA_SEIO a handle for the program is a path name of standard input/output file
 TMA_LINK a handle for the program is a link to the file of the standard file system
 TMA_PTR a handle for the program is a pointer to the codes loaded in memory

Return Parameter

| | | | |
|----|------|------|---------------------------------------|
| ER | ercd | >= 0 | normal completion (system program ID) |
| | | < 0 | error code |

Error Code

| | |
|--------|--|
| E_FACV | no access privileges (E) for the file (when TMA_SEIO, TMA_LINK is specified) |
| E_MACV | access to address (info, hdr (when TMA_PTR is specified)) not allowed |

| | |
|---------|--|
| E_BUSY | could not open the file because it is already opened exclusively |
| E_IO | input/output error occurred |
| E_NOEXS | file does not exist |
| E_NOFS | the file system to which the file belongs is not connected |
| E_NOMEM | insufficient memory area (insufficient memory area to load) |
| E_REC | no program record present in the file. Or the content of program record is unusual (when TMA_LINK is specified). |

Description

Loads the program code to the system memory space as a system program, and allocates a unique system program ID.

spgatr of T_LSPG structure indicates the attribute of a created process.

If TMA_SEIO attribute is specified, the content of the specified file is loaded as a program code. Specify the path name of the standard input/output of the target file for spghdr.

If TMA_LINK attribute is specified, the content of the first executable program record in the file of the specified standard file system is loaded as program codes. Specify the link (LINK*) to the standard file system file for spghdr.

If TMA_PTR attribute is specified, program codes in memory may be set to system program. Specify the pointer of the program codes in memory for spghdr. Note that the format of the program codes in memory and the running methods are implementation-dependent.

If load is successful, loaded start address is returned to info[0], and loaded last address is returned to info[1].

System programs are simply loaded (mapped) on memory only, and relocation of the symbol address, etc. is not processed. Moreover, when the specified system program is the same as the one which has already been loaded, a different new memory space is allocated and loading is conducted. In this case, a different system program ID is allocated respectively.

The area where the system program was loaded is always made resident.

Unload System Programs

tkse_unl_spg

C Language Interface

```
ER ercd = tkse_unl_spg(W progid);
```

Parameter

| | | |
|---|--------|-------------------|
| W | progid | system program ID |
|---|--------|-------------------|

Return Parameter

| | | |
|----|------|------------|
| ER | ercd | error code |
|----|------|------------|

Error Code

| | |
|------|-------------------------------------|
| E_OK | normal completion |
| E_ID | no system program (progid) existent |

Description

Unload the loaded system program specified by “progid”. For all areas which were mapped in memory for the system program, map release is conducted. Whether or not the system program is being used is not a concern.

Get Version

tkse_get_ver

C Language Interface

```
ER ercd = tkse_get_ver(T_VER* version);
```

Parameter

T_VER *version storage area of version

```
typedef struct {
    UH        maker;        /* maker*/
    UH        id;           /* style number */
    UH        spver;        /* specification version */
    UH        prver;        /* product version */
    UH        prno[4];      /* product management information */
    UH        cpu;          /* CPU information */
    UH        var;          /* variation descriptor */
} T_VER;
```

Return Parameter

ER ercd error code

Error Code

E_OK normal completion
E_MACV access to inaccessible address (version) not allowed

Description

Acquires AMP TKSE version information and stores it in version.

4.14 Shared Library Function

4.14.1 Shared Library Function Overview

The shared library function of AMP TKSE manages the program codes (libraries) shared from multiple processes loaded at runtime.

Shared libraries are present as shared library files in file system, and are loaded using a library function at program's runtime, and become available after symbols resolution.

The loading and symbols resolution of shared libraries are performed by a function call provided as a library. The shared libraries also allow you to load and resolve symbols automatically by placing them on shared library path specified when building the system.

The features of both methods mentioned above are as follows:

(1) Available using library function from user program

- Load and symbols resolution explicitly call library function
- Shared library files can be placed anywhere in the system

(2) Automatically used when process is created

- Loading is done automatically when process is created
- Symbols resolution is done automatically at runtime
- Shared library files should be placed on the path specified when building the system

Shared library function depends on language processor functions such as compiler, linker. To create shared library files, language processor should have the function to create position independent codes.

The API specification of the common library management function is the same as in the T-Kernel Standard Extension Version 1.00 Specification.

4.14.2 Library Call

Open Shared Library

dlopen

C Language Interface

```
void *handle = dlopen(const char *filename, int flag);
```

Parameter

| | | |
|-------------|----------|-----------------------------------|
| const char* | filename | path name of shared library files |
| int | flag | symbols resolution setting |

(RTLD_LAZY || RTLD_NOW) | [RTLD_GLOBAL]

| | |
|---------------------|---|
| RTLD_LAZY (0x01) | resolve ambiguous symbols in order at runtime |
| RTLD_NOW (0x02) | resolve all ambiguous symbols at loading time |
| RTLD_GLOBAL (0x100) | set symbols to global |

Return Parameter

| | | | |
|--------|--------|-----|--|
| void * | handle | > 0 | normal completion (handle to shared library) |
| | | = 0 | error exit |

Description

Loads shared library in the path specified by filename to the local space of this process.

The path name follows the path name specification of standard input/output function.

If loading is successful, a handle to shared library (>0) is returned as a return value. If path name is NULL pointer, shared library is not loaded and a handle to main program is returned.

The retrieved handle is used as an argument to dlsym().

The symbols resolution is set by specifying either RTLD_LAZY or RTLD_NOW to flag. At the same time, RTLD_GLOBAL is also set by taking the logical union (OR).

If RTLD_NOW is specified, dlopen() returns after resolving all the undefined symbols in libraries. If unsuccessful, an error will be returned.

If RTLD_LAZY is specified, symbols values are resolved the first time they are required at runtime. The operation is not ensured when ambiguous symbols are not resolved (normally, exception occurs).

If RTLD_GLOBAL is specified, external symbols of loaded shared library can be used to resolve symbols of other shared libraries opened afterward.

Find Symbol of Shared Library

dlsym

C Language Interface

```
void *val = dlsym(void *handle, const char *symbol);
```

Parameter

| | | |
|--------------|--------|--------------------------|
| void * | handle | handle to shared library |
| const char * | symbol | specified symbol |

Return Parameter

| | | |
|--------|-----|--|
| void * | val | != NULL normal completion (symbol value) |
| | | = NULL error |

Description

Finds the symbol specified by symbol from shared library specified by handle and returns the value.

If symbol is not found, NULL will be returned.

Special handles such as RTLD_NEXT, RTLD_DEFAULT can be set to handle.

If RTLD_NEXT is set to handle, symbols search begins with "next" shared library after the shared library which called dlsym().

If RTLD_DEFAULT is set to handle, symbols search is done in the scope of the shared library which called dlsym().

Close Shared Library

dlclose

C Language Interface

```
int rtn = dlclose( void *handle );
```

Parameter

| | | |
|--------|--------|--------------------------|
| void * | handle | handle to shared library |
|--------|--------|--------------------------|

Return Parameter

| | | | |
|-----|-----|-----|-------------------|
| int | rtn | = 0 | normal completion |
| | | < 0 | error |

Description

Closes the shared library specified by handle.

If the shared library was called by dlopen() multiple times, it is at last closed after it was called by dlclose() that many times.

If shared library is closed, symbols in the library will become unavailable.

Retrieve Symbol Information of Shared Library

dladdr

C Language Interface

```
int rtn = dladdr( void *addr, Dl_info *info );
```

Parameter

| | | |
|-----------|------|------------------------------|
| void * | addr | symbol's address |
| Dl_info * | info | specified symbol information |

```
typedef struct {
    const char    *dli_fname;    /* file name */
    void          *dli_fbase;    /* base address */
    const char    *dli_sname;    /* symbol name */
    void          *dli_saddr;    /* symbol's address */
} Dl_info;
```

Return Parameter

| | | | |
|-----|-----|------|-------------------|
| int | rtn | = 0 | error |
| | | != 0 | normal completion |

Description

If the address specified by `addr` is inside the one of shared libraries, symbol information at the address is returned to `info`.

The pointer to the file name of shared library is stored to `dli_fname`. The file name convention complies with standard input/output specification.

The base address (load offset) of shared library is stored to `dli_fbase`. This is used as an argument to `dlsym()` as a handle to shared library.

The pointer to the name of closest symbol is stored to `dli_sname` with the same or smaller value specified by `addr`. The symbol value (address) of `dli_sname` is stored to `dli_saddr`.

5. Implementation Method

5.1 Overview

AMP TKSE is a function extension program for AMP T-Kernel, and its functions are implemented as AMP T-Kernel subsystems. The following shows a list of subsystems used by AMP TKSE.

- Memory management subsystem
- Segment management subsystem
- Process/Task management subsystem
- Interprocess message subsystem
- Global name subsystem
- Intertask synchronization and communication subsystem
- Standard input/output subsystem
- Standard file management subsystem
- Filesystem Sharing Management Subsystem
- Event management subsystem
- Device management subsystem
- Time management subsystem
- Object Management Subsystem

An application invokes a AMP TKSE function using a system call (`tkse_xxx_yyy`) implemented as the subsystem extended SVC. This system call is usually invoked via an interface library linked to the application.

5.2 Memory Management and Segment Management

The memory space of AMP TKSE is managed by two subsystems: Memory management subsystem and segment management subsystem.

The memory management subsystem executes block-by-block memory area management.

When a memory area is allocated, each of the memory blocks making up this area is registered to a page table. A page table is a data structure used to retain associations between logical and physical addresses and various attributes of memory blocks. MMUs realize conversions between logical and physical addresses and restriction on access to memory areas using the information in a page table. A page table, existing independently for each process, is initialized at process creation and discarded at process exit.

The segment management subsystem conducts virtual memory and memory space management such as mapping of a memory space to a disk or setting of resident/non-resident attributes.

If the physical memory runs short while virtual memory is enabled, segment management writes out a memory page currently not in use to a page file, discards the memory page (page-out), and allocates it as a new memory area. If there is memory access to a memory page that has been paged out, segment management reads this memory page into the memory (page-in) and executes the access.

The page-in and page-out processes are executed in a task context. While the task-independent portion is executed, therefore, the physical memory must not run short or the memory area that has been paged out must not

be accessed. The codes that run as the task-independent portion and the data that it references while running must be located in resident memory.

5.3 Process/Task Management

(1) Memory space

The local memory space of a process is implemented as a task space of AMP T-Kernel. All the processes have an independent task space, and the tasks in a process belong to the task space of this process.

(2) Resource group

Each process has an independent resource group. A new resource group is created at process creation and deleted at process exit.

A resource group is used to keep information unique to its process such as process management information, file descriptor, and current directory information.

(3) Task protection level

A task in a user process runs at protection level 3 and a task in a system process runs at protection level 1. These processes cannot access the system area used by drivers and the OS (area at protection level 0). As an exception, the initial process runs at protection level 0. Therefore, the initial process can access the system area.

(4) Task priority allocation and scheduling

The task priority (*sepri*) of AMP TKSE is allocated to the AMP T-Kernel task priority (*kpri*) as follows:

[Table4] Task priority allocation

| AMP TKSE task priority ($sepri$) | AMP T-Kernel task priority ($kpri$) | Priority assignment method | Slice time (milliseconds) |
|---------------------------------------|---------------------------------------|----------------------------|---------------------------|
| – | 1 to 7 | Not assigned | – |
| 0 to 127 (Absolute priority group) | 8 to 135 | $kpri = sepri + 8$ | 10 (Fixed value) |
| – | 136 to 137 | Not assigned | – |
| 128 to 191 (Round robin group 1) | 138 | $kpri = 138$ (Fixed value) | $192 - sepri$ |
| 192 to 255 (Round robin group 2) | 139 | $kpri = 139$ (Fixed value) | $256 - sepri$ |
| – | 140 | Not assigned | – |

For a task in the absolute priority group, the sum of the AMP TKSE task priority and the offset value (plus eight) is assigned to the AMP T-Kernel task priority. Since relative relationship of priorities is preserved, a task in the absolute priority group is scheduled in the same order of priorities as in AMP T-Kernel. If there are multiple tasks with the same priority, AMP TKSE schedules them equally in a round robin fashion at fixed intervals (10 msec), but AMP T-Kernel does not automatically pass the execution privilege among them unless the slice time of a task is explicitly set.

All the tasks in round robin group 1 are allocated to the AMP T-Kernel task priority of 138 regardless of the AMP TKSE task priority. The round robin algorithm is realized by setting the task time slice according to the AMP TKSE task priority and by changing precedences the same AMP T-Kernel task priorities. The task slice time is as follows; (slice time) = $192 - (\text{AMP TKSE task priority})$.

Likewise, all the tasks in round robin group 2 are set to the AMP T-Kernel task priority of 139. The round robin algorithm is realized in the same way as in round robin group 1. The task slice time is as follows; (slice time) = $256 - (\text{AMP TKSE task priority})$.

AMP T-Kernel task priorities (1 to 7, 136 and 137, and 140) not allocated as the AMP TKSE task priorities are not available in AMP TKSE. However, AMP T-Kernel applications can freely use these priorities.

5.4 Interprocess Messages

The content of a message that has been sent is copied to an area of system memory space reserved by the sender and is inserted to the message queue of a destination process. The inserted data is copied again to the buffer area specified at message reception.

The synchronization at message transmission and reception is realized by a task event. Task event number 1 is used to wait for message reception when CONFM was specified at message transmission. Task event number 2 is used to release the wait state using `tkse_brk_msg()`. Software other than message management must not send these task event numbers to any task in the process.

To use a message handler, it is necessary to realize asynchronous reception of a message by raising a task exception in the main task of the process in which the message handler has been registered. Software other than message management must not raise a task exception in the main task in a process. In the initial process that runs at protection level 0, no task exception can be raised due to a restriction on AMP T-Kernel. Therefore, asynchronous message reception using a message handler cannot be executed.

5.5 Intertask Synchronization and Communication Functions

The intertask synchronization and communication functions are realized by indirectly invoking a relevant system call of AMP T-Kernel.

An extended function of AMP TKSE is automatic object deletion at process exit (specification of `TA_DELEXIT`). To realize this function, each process has the list information of objects that have been created as a resource group. Furthermore, this list information is associated with each object using the extended information `exinf` of the object. Therefore, the AMP TKSE intertask synchronization and communication functions cannot use the extended information `exinf`.

5.6 Device Management Function

The device management functions are realized by indirectly invoking the device management functions of AMP T-Kernel/SM. However, devices cannot be registered or deleted from AMP TKSE.

5.7 Time Management Function

The time management functions are implemented using the AMP T-Kernel time management functions. System time is initialized by reading the RTC time using the clock driver when AMP TKSE is started. If there is no clock driver or the RTC time is invalid, the system time is undefined.

5.8 Object Management

Object management is implemented by using the domain function of AMP T-Kernel.

The kernel domain of AMP TKSE is implemented as a kernel domain of AMP T-Kernel. Both are same in the implementation.

The process domain is created as a domain of the public attribute that belongs to the kernel domain when the process is created. When the process is deleted, the process domain is deleted.

For the synchronization and communication object, the domain that the object belongs to and access protection attribute of AMP T-Kernel are decided according to the access attribute when the object is created. The table below shows the correspondence.

| AMP TKSE Access Attributes | Domain that the Object Belongs to | AMP T-Kernel Access Protection Attribute |
|----------------------------|-----------------------------------|--|
| Global Attribute | Kernel Domain | Public Attribute |
| Kernel Local Attribute | Kernel Domain | Protect Attribute |
| Process Local Attribute | Process Domain | Private Attribute |

Although tasks belong to the process domain and act as private attributes on AMP TKSE, the access protection of AMP T-Kernel is specified for the public attribute. Protection against access for tasks from other processes is realized by AMP TKSE. This is due to the fact that access by tasks from other processes must be possible since task events and task exceptions are used for messages between processes.

- Global name
 - N: GlobalNameLimit Maximum number of global names

- Time management
 - N: CmClkUpd Clock update notification (0: Enabled, 1: Disabled)
 - Issues a clock update notification at zero second every minute. A notification is issued by invoking `tkse_brk_msg`.

- Standard file management
 - N: FmTskPri File management task priority (common task)
 - File management task priority (task of each file system)
 - N: MaxOpenF Maximum number of files that can be opened simultaneously
 - N: SyncTimeOut Synchronization timeout time (in milliseconds)
 - N: FmTimeStamp Time stamp update control (0: Updated, 1: Not updated)
 - Control whether or not to update the time stamp when reading a record.
 - Update when set to 0. Don't update when set to 1.

- Event management
 - N: EmTskPri Device event reception task priority
 - N: MaxEvtQ Event queue size (in bytes)
 - N: MaxEvMsg Maximum number of registrations of event message
 - N: EvtLife Event lifetime (in milliseconds)
 - Set the lifetime of a wheel event.

- Standard input/output
 - N: UxMaxOpenF Maximum number of files that can be opened simultaneously (in each process)
 - N: UxFsTskPri File system task priority
 - N: UxSyncTimeOut Synchronization timeout time (in milliseconds)